NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPROVISED EXPLOSIVE DEVISE PLACEMENT DETECTION FROM A SEMI-AUTONOMOUS GROUND VEHICLE

by

Benjamin D. Miller

December 2006

Thesis Advisor: Richard Harkins
Second Reader: Nancy Haegel

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE Improvised Explosive Devise Placement Detection from a Semi-Autonomous Ground Vehicle	**5. FUNDING NUMBERS**
6. AUTHOR(S) Benjamin D. Miller	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Improvised Explosive Devices (IED's) continue to kill and seriously injure military members throughout the Iraqi theatre. Autonomous Ground Vehicle (AGV) seeks to identify the human presence placing the IED and then report that contact to a unit of action. This research developed a semi-autonomous platform that can navigate to waypoints, avoid obstacles, investigate possible threats and then detect motion that triggers a visual camera. The information is then relayed back to the user and can trigger a variety of actions. AGV has been tested in a numerous environments with a wide range of success. It is limited by the communication range from its standard 802.11G router and the continuous availability of the global positioning system. Terrain with extensive peaks and valleys is not ideal for the current platform. However, for detecting the human presence that is consistent with IED placement, AGV is well suited.

14. SUBJECT TERMS Robotics, Autonomous, Human Presence, Improvised Explosive Device	**15. NUMBER OF PAGES** 111
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IMPROVISED EXPLOSIVE DEVISE PLACEMENT DETECTION FROM A SEMI-AUTONOMOUS GROUND VEHICLE

Benjamin D. Miller
Major, United States Army
B.S., United States Military Academy, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
December 2006

Author: Benjamin D. Miller

Approved by: Richard Harkins
Thesis Advisor

Nancy Haegel
Thesis Co-Advisor

James Luscombe
Chairman, Department of Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Improvised Explosive Devices (IEDs) continue to kill and seriously injure military members throughout the Iraqi theatre. Autonomous Ground Vehicle (AGV) seeks to identify the human presence placing the IED and then report that contact to a unit of action. This research developed a semi-autonomous platform that can navigate to waypoints, avoid obstacles, investigate possible threats and then detect motion that triggers a visual camera. The information is then relayed back to the user and can trigger a variety of actions. AGV has been tested in a numerous environments with a wide range of success. It is limited by the communication range from its standard 802.11G router and the continuous availability of the global positioning system. Terrain with extensive peaks and valleys is not ideal for the current platform. However, for detecting the human presence that is consistent with IED placement, AGV is well suited.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF EQUATIONS

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Thanks to LT Jason Ward for his previous work with the JAVA programming. In addition, ENS Tom Dunbar's extensive work on the PWM circuit was a great starting point for AGV's motor functions. Without the help of George Jaksha, AGV would never have taken its current form. Thanks to the lab group: LT Andy Hoffman, LT John Herkamp, and ENS Todd Williamson for all their advice and support. Finally, thanks to my thesis advisor Professor Richard Harkins for allowing me to see this project through regardless of the amount of destroyed components.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The military has a wide range of projects underway to deploy robotic platforms in operational environments. These projects impact the battlefield with varying degrees of success. There are two distinct areas in which the military currently employs robotic platforms. Most notably is the Unmanned Aerial Vehicle (UAV). The other is the use of a ground robotic platform for IED (Improvised Explosive Device) detection and destruction. Neither of these platforms is autonomous. Currently, Explosive Ordnance Disposal (EOD) units use the TALON platform to aid them in completion of their missions.

The Department of Defense has a Joint Robotics Program (JRP). This program is designed to research unmanned ground systems for use in a wide range of military applications. The program started by examining the performance of different platforms over diverse terrain. As the program evolved, more autonomous functions were investigated. As with all technical based programs, the end stage product is limited by the available technology [Ref. 20]. Currently, most participants have encountered similar problems. Most noticeably are onboard sensor limitations and the processing power of the onboard computers.

A. UAV

The most common and unclassified UAV program is the Hunter project [Ref. 1]. The system is capable of the following missions: real time imagery intelligence, artillery adjustment, battle damage assessment,

reconnaissance and surveillance, target acquisition and battlefield observation. Figure 1 is a common Hunter employed on a military mission. Figure 2 shows the Army personnel controlling the UAV. The control station consists of a separate vehicle and an extensive amount of electronics.

Figure 1. A typical UAV on a mission (From [Ref. 1]).

Figure 2. The control station for the UAV (From [Ref. 1]).

B. TALON

Although there are specialized ground robotic platforms that the Army employs (cave searching in Afghanistan, room clearing in urban environments, etc.), the most commonly used ground robot is the TALON platform. Figure 3 shows a typical EOD mission for the TALON.

There is testing underway to employ this platform in a variety of roles, but the only current mission is with the EOD units for IED detection and disarmament. The TALON is not autonomous, weighs approximately 100 lbs, costs well over $50,000, and has dimensions of approximately 3′ x 2′ x 3′ [Ref. 2]. These factors make it unsuitable for use in the proposed missions for AGV.

Figure 3. The TALON robot employed to investigate an IED (From [Ref. 2]).

C. PREVIOUS NAVAL POSTGRADUATE SCHOOL (NPS) PROJECTS

The Small Robot Technology (SMART) initiative at NPS develops prototype robotic platforms for the military. The robots' missions are extensive and they range in size from inches to several yards. The development for the current AGV started with a prototype known as Bender. Bender was not intended for a specific mission, but was designed to investigate autonomous architecture. It had a hardened track chassis with a box shape. It incorporated ultrasound sensors to aid in collision avoidance. The basic programming (Dynamic C) along with the on-board computer (a commercial BL2000) is the same as AGV uses. Bender had a web-cam to view any contacts that came within its path. It functioned autonomously to the point where it could move from one point to another while avoiding large obstacles. Figure 4 shows Bender in its final form. The size, slow speed and cumbersome movement made the platform unsuitable for our AGV's proposed mission.

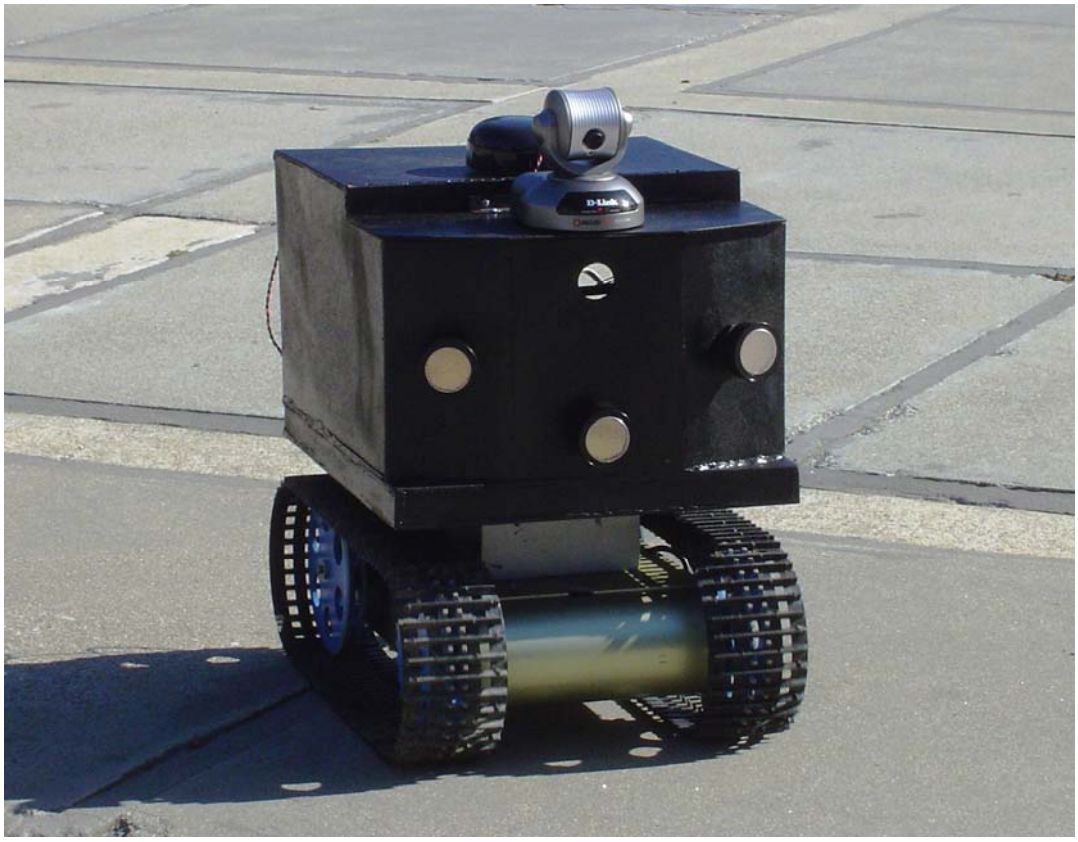


Figure 4. Bender in its final form in the NPS quad.

The next generation of autonomous robots turned towards naval specific applications. LT Jason Ward created Lopez. This platform was the first prototype for a surf-zone robot that can conduct reconnaissance and surveillance on the beachhead. In the future, these platforms will be launched from surface ships or submarines. Figure 5 shows the working Lopez model.

The third generation of the robots from the SMART program was created in collaboration with ENS Tom Dunbar from NPS and Case Western University. Agbot is a much more powerful aluminum version of Lopez. Although this platform had problems with its structure, it is a working prototype that has been successfully tested on sand, grass, and

concrete [Ref. 3]. Figure 6 shows Agbot prepared for a test run. Both Lopez and Agbot were designed to run autonomously from a Java interface. Although Lopez and Agbot are for naval applications, the same basic components (GPS, on board computer, compass, camera, and router) along with the computer coding are incorporated on AGV.

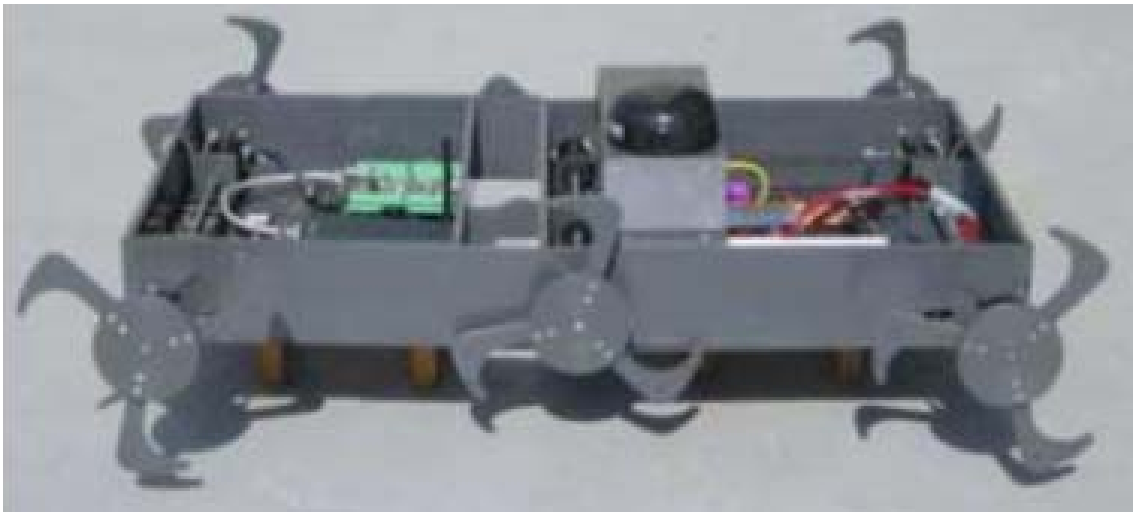


Figure 5. The figure shows Lopez with all of its components installed (From [Ref. 3]).

Figure 6. The figure shows Agbot with all of its components installed (From [Ref. 3].

D. AGV MOTIVATION

The United States Army currently has soldiers being killed or seriously wounded at an alarming rate while deployed in Iraq. Based on my experiences, the IED accounts for the majority of those casualties. To date, no effective techniques have been found to adequately guard the thousands of miles of main supply routes that are needed to re-supply units and conduct continuing missions. As an example, my company was assigned well over 50 miles of roadways to continually prevent IED placement. In addition to that continuing mission, we had Quick Reaction Force duty, periodic raids, base security, and a host of other tasks. To adequately guard the roadways was virtually impossible given my 6 M1A1 tanks, 8 BFV (Bradley Fighting Vehicles) and 135 soldiers. Within the Task Force, my company had the only heavy armor. The IED threat made the risk of putting softer skinned vehicles continuously in sector too great.

The insurgents place IEDs in three stages:

1. The hole is dug or camouflage created and the IED is placed,
2. The detonation device is placed,
3. The connections are made.

Detecting the IED itself is extremely difficult. However, in each of the three placement stages, there is a human presence that can be detected. The tank or BFV in that sector has to be watching that specific small area at the exact right time to detect the insurgents placing the IED. The area is just too vast and the insurgents are fairly intelligent. If a tank or BFV moves from one location, the insurgent knows he has a window of time when that area is

not being watched, a perfect time to do one of the three IED placement stages. I estimate that 40% – 50% of the IED casualties occur in a location where a previous IED was detonated. There are many reasons for this, but if nothing else the insurgents know a pattern that works in that location and it is very difficult to stop their cycle. AGVs could be tactically deployed to those areas to detect any suspicious activity.

II. ROBOTIC FUNCTIONAL DESIGN

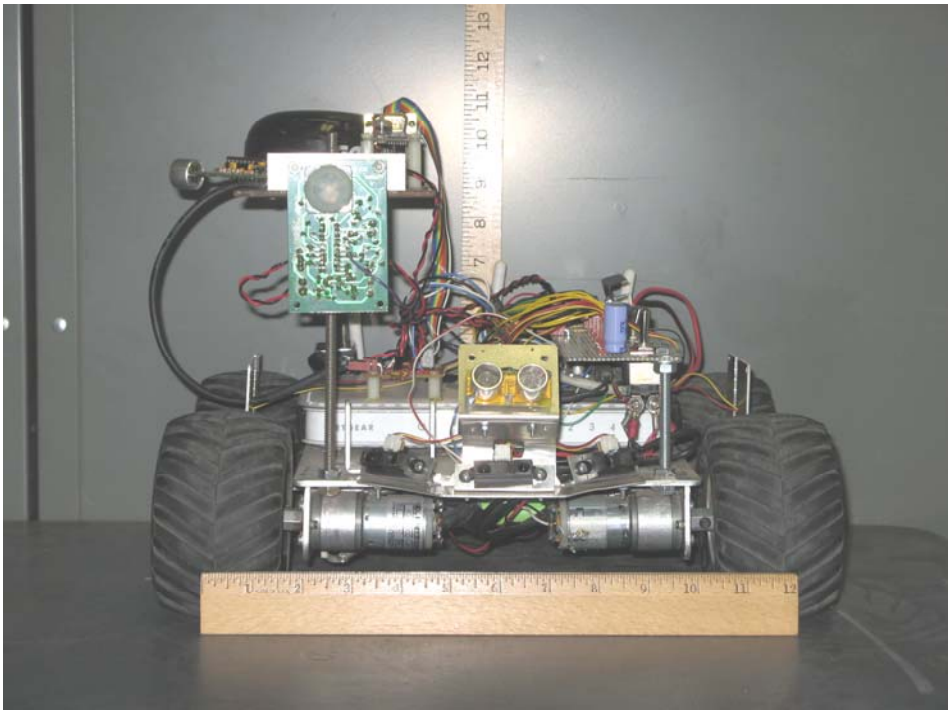


Figure 7. AGV seen from the front.

A. PROBLEM SOLUTION

Figure 7 is the AGV designed to combat the problem listed above in the project motivation. AGVs can be tactically placed to observe potential IED locations at appropriate times. Ideally the entire country or at least the main supply routes would be over watched with hard mounted visual and thermal cameras. The power would come directly from the Iraqi power grid and each camera would have a back up battery. This is not feasible given the current operational and tactical situation. AGVs can be used in a number of tactical scenarios, individually or in groups. When an armored vehicle moves from an area a crewmember could easily place a single AGV to monitor the general location just vacated. In daylight, the robot

could be disguised within the tremendous amount of trash lining all of the main roads in Iraq. At night, very little disguise would be necessary.

AGV is equipped with acoustic and IR (infrared) motion detectors. These sensors can trigger a thermal or visual camera once they detect motion. Upon detection, AGV sends a message back to the unit. It can then be turned over to manual control and the person monitoring can reposition (if necessary) the camera to decide if an IED is being emplaced. Or, AGV can continue to send video feed and text messages in autonomous mode. Ideally, presence of insurgents with IED material would trigger an armored or infantry response for action. If soldiers in the area could not move to intercept the insurgent in time for a possible kill, an ambush could be employed for the next stage of emplacement. At a minimum they would secure the area for EOD personnel to come and further evaluate the ordinance.

A second technique could be to place a group of AGVs in habitual problem areas. As an example, five AGVs networked together could over watch a given space of MSR. There are a number of scenarios to tactically employ AGVs, but once in sector the odds of observing the IED emplacement have increased significantly. If one AGV detects a possible emplacement, the other four could be repositioned to look for the building the insurgents were using and the routes they took to get in and out. This may not allow for the instantaneous kill of the insurgents emplacing the IED, but it would certainly prevent a convoy from moving through the possible IED location.

B. BLOCK DIAGRAM

Figure 8 is shows the entire AGV process from when it is initially placed until it detects the human presence placing the IED. As a summary, once dropped the user needs to decide if AGV needs to move to a different location. This can be accomplished autonomously by sending AGV to a waypoint, or by manually controlling it into position. If moved autonomously AGV will internally calculate the heading it needs to travel on and then rely on the GPS to stop at the appropriate location. While moving autonomously, the Proportional Integral Derivative (PID) control will ensure it stays on the appropriate heading. Once at the desired location, AGV will turn off its motors and turn on the detection (consisting of the motion detectors). Once a presence (or motion) is detected AGV will send a text message back to the user. The user then has the option of continuing to just monitor the motion, or to activate the on board camera to take a snap shot photos (automatically sent to the GUI) of the area. If no presence is detected, AGV will continue to monitor but has the ability to autonomously move through a predetermined set of waypoints (detecting at each) or the user can always manually control AGV to a different location. Each time AGV moves, the detection will be turned off and reset.

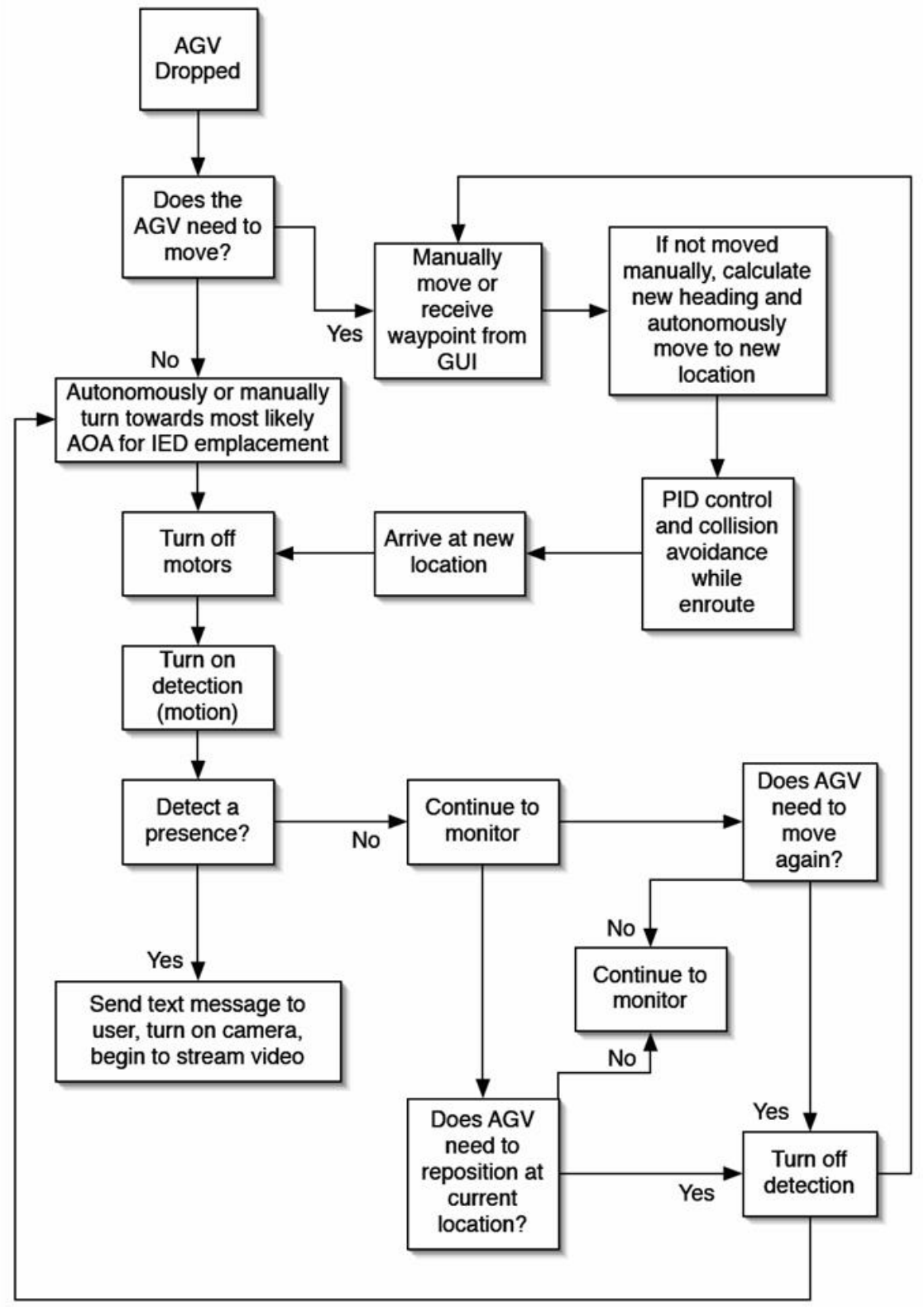


Figure 8. The diagram depicts the typical employment of AGV.

III. EXPERIMENTAL DESIGN

Table 1 shows a summary of the all the main components AGV uses.

Hardware Component	Vendor	Price	Operating Parameters
AGV Base	Superdroid Robots	$209.35	14" x 13" x 4", 1" ground clearance
Batterey	Superdroid Robots	$34.70	NiMH, 2 x 10 array of AA batteries
Motors	Superdroid Robots	$18.95 ea.	24-volt, 195-rpm gear motors
Motor Controllers	Superdroid Robots	$26.95 ea.	12-55VDC, 3A to 6A, 4 pin (brake, direction, PWM, ground)
Ultrasonic Range Finder	Superdroid Robots	$62.00	Objects from 0" to 254", utilizes IIC/I2C bus
IR Rangers	Superdroid Robots	$14.85 ea.	Objects from 5cm to 80cm, analog output
PIR Motion Detector	Willy's Electronics	$36.69	Motion up to 25m, infrared input, utilizes a Fresnel Lens
Ultrasonic Motion Detector	Kitsrus	$34.99 apx.	Motion up to 10m, 40 kHz frequency
Router	Newegg	$129.99	802.11GHz wireless router, max range apx. 300m
BL2000	Rabbit	$449.00	Single-board computer, 22.1 MHz, 11 analog inputs, 2 analog outputs
Compass	Honeywell	$675.00	Digital magnetic compass, heading to +/- 0.5 degrees
GPS	GPS City	$174.95	Low voltage system, utilizes WAAS network
Camera	D-Link	$94.99	Web server 10/100Mbps, 640x480, 320x240 resolution

Table 1. The table is a summary of the main hardware components AGV uses.

A. PLATFORM

1. AGV Base

The base of the AGV is an aluminum welded base approximately 8.5 by 10 inches. The sides are 1.75 inches high. The wheels are four inches in diameter and are presently filled with standard packing foam for stability. With the wheels mounted, AGV is 14 by 13 inches. It has

approximately one inch of ground clearance, but with the collision avoidance it can traverse a surprising wide range of terrain [Ref. 4]. Figure 9 shows the bottom of AGV along with the wheelbase.

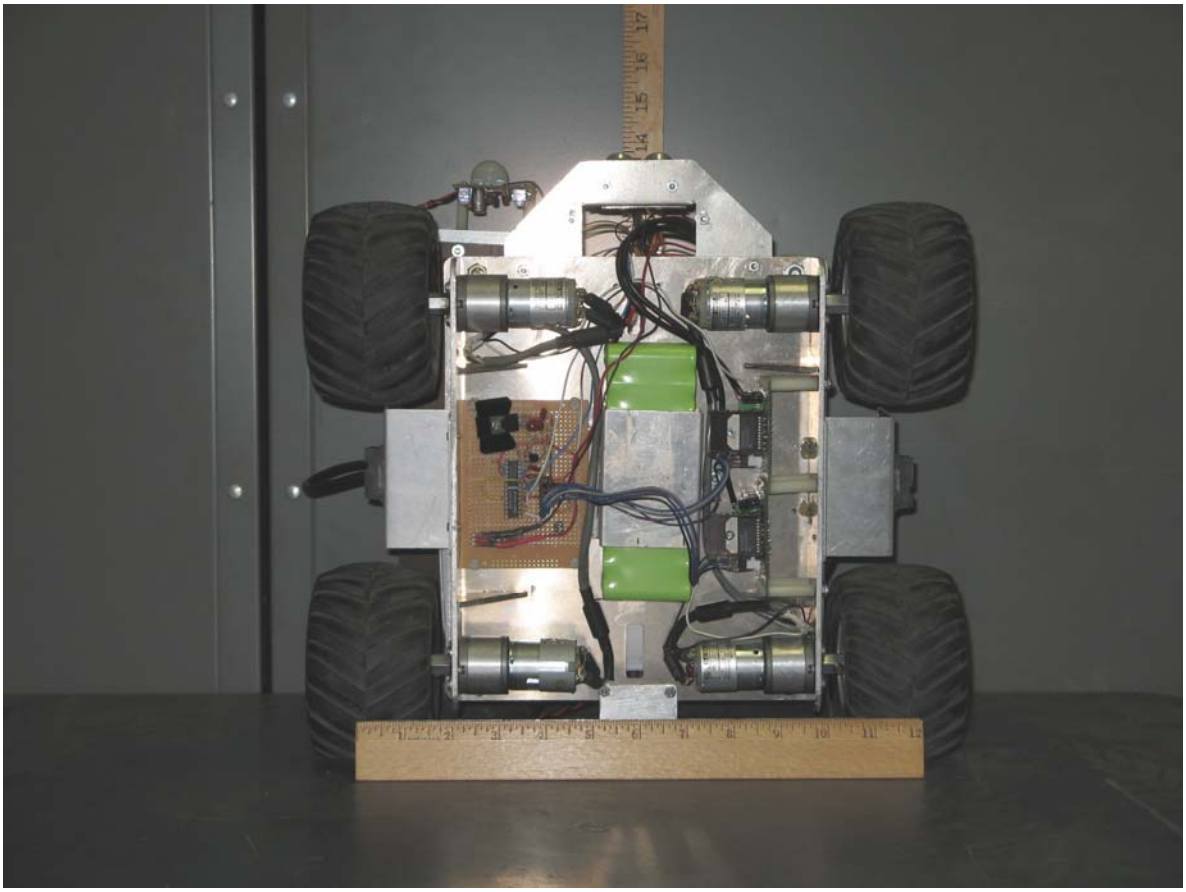


Figure 9. AGV viewed from the bottom.

2. Power Bus and Battery

The power bus with battery consists of all commercial off the shelf (COTS) products. The battery (Figure 10) is mounted on the underside of AGV. It is a 20 cell, 24VDC 2000mAhr rechargeable Nickel Metal Hydride (NiMH) battery pack [Ref. 5]. During continuous operations on a full charge, the battery provides approximately two hours of on station time. As an aside, the motion detectors are

powered by a separate 9-volt battery for reasons discussed later. The battery pack has a 2x10 array of standard AA batteries.

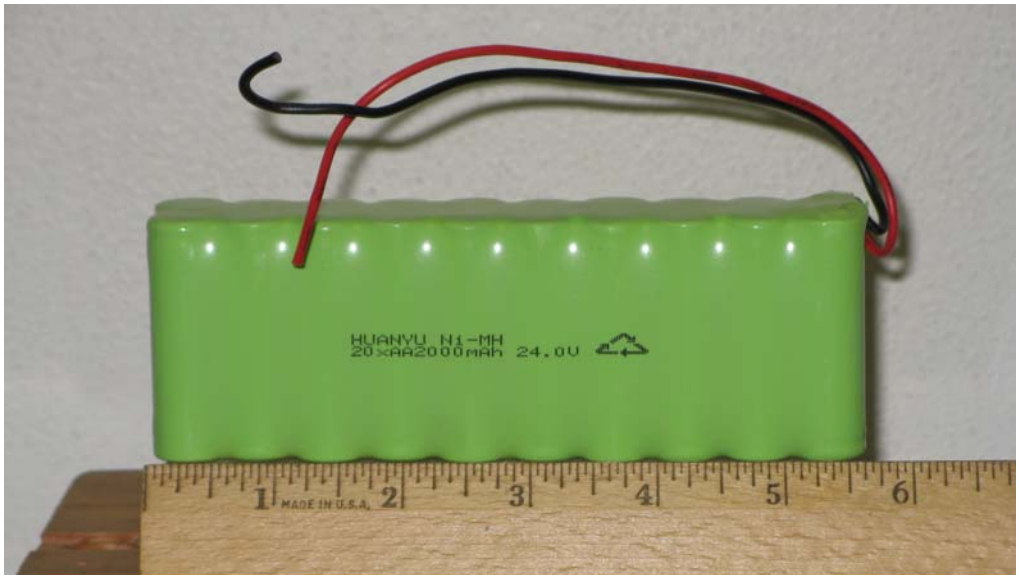


Figure 10. 20 cell, 24 VDC, 2000 mAhr, rechargeable NiMH battery pack.

The battery connects with the master switch that then connects to a standard 12-volt regulator rated for 5 amps of current [Ref. 6]. The 12-volt regulator has two outputs that send voltage to four ports for the components that require 12 volts (GPS, compass, router and the BL2000). The other output feeds into a standard ML7805 5-volt regulator. The 5-volt regulator has eight ports for the various components that require 5-volts (IR rangers, sonar, etc.). The power bus has connectors for the components that require IIC (computer coding discussed later) in order to function. There is a 16-volt 1000-microfarad electrolytic capacitor protecting the 5-volt ports. Additionally, the bus has a cut off switch for the motors.

The switch either turns the motors completely off or allows the BL2000 to control them. Figure 11 shows the complete power bus.

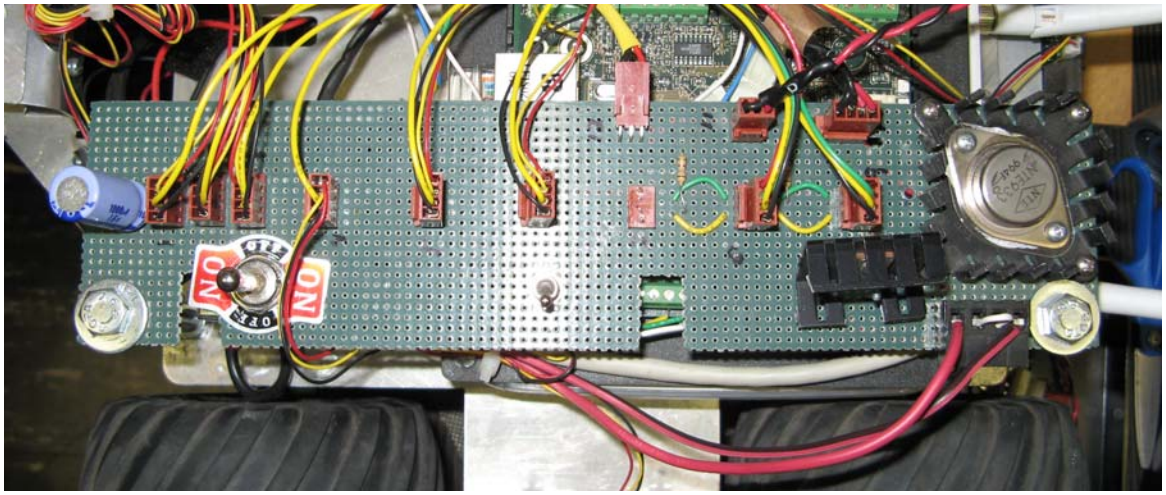


Figure 11. Power Bus.

3. Motors and Motor Controllers

The motors are not powered through the power bus on the topside of AGV. Their connections run directly from the battery to the motor controllers through a circuit located on the underside of the robot. Each wheel has its own motor (see Figure 12) and each side of the robot is controlled by one motor controller (see Figure 13). The four motors are 24-volt, 195-rpm gear motors. They are rated for less than 150-mA of current when loaded and have a torque rating of 1.4 kgf-cm [Ref. 7].

Figure 12. One of the four motors connected to each wheel.

The motor controllers are simple control boards (see Figure 13). They are designed to work specifically with the IG32 motors for the AGV platform [Ref. 8].

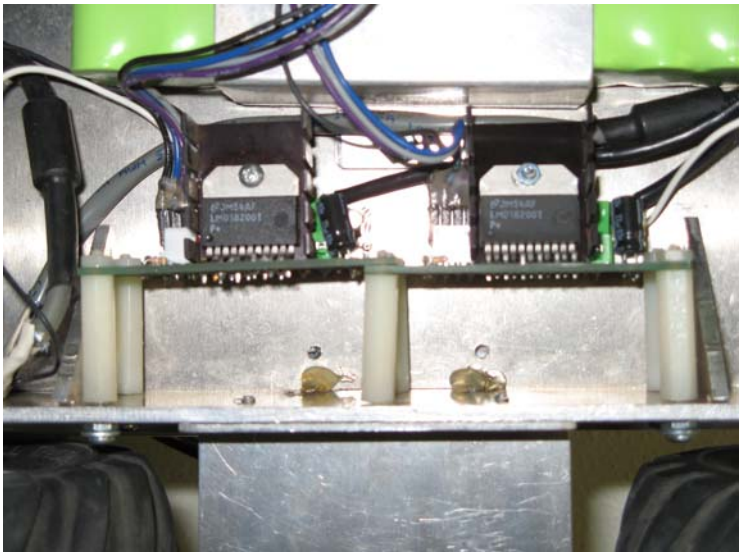


Figure 13. Both motor controllers mounted on the underside of AGV.

The boards are easy to construct and have accessible external pins for the direction, brake, and the pulse width

modulation (PWM) option. AGV does not use the available PWM portion on the motor controller. Instead it has an externally constructed PWM circuit (discussed later) in order to better control the speed. In the current configuration pin P (PWM) is tied high (5 volts), pin B (brake) is controlled from the BL2000. On is high and off is low (ground). Pin D (direction) is controlled through the external PWM circuit. It receives a standard square wave that controls the speed depending on the duty cycle. Figure 14 is a functional diagram for the entire circuit board.

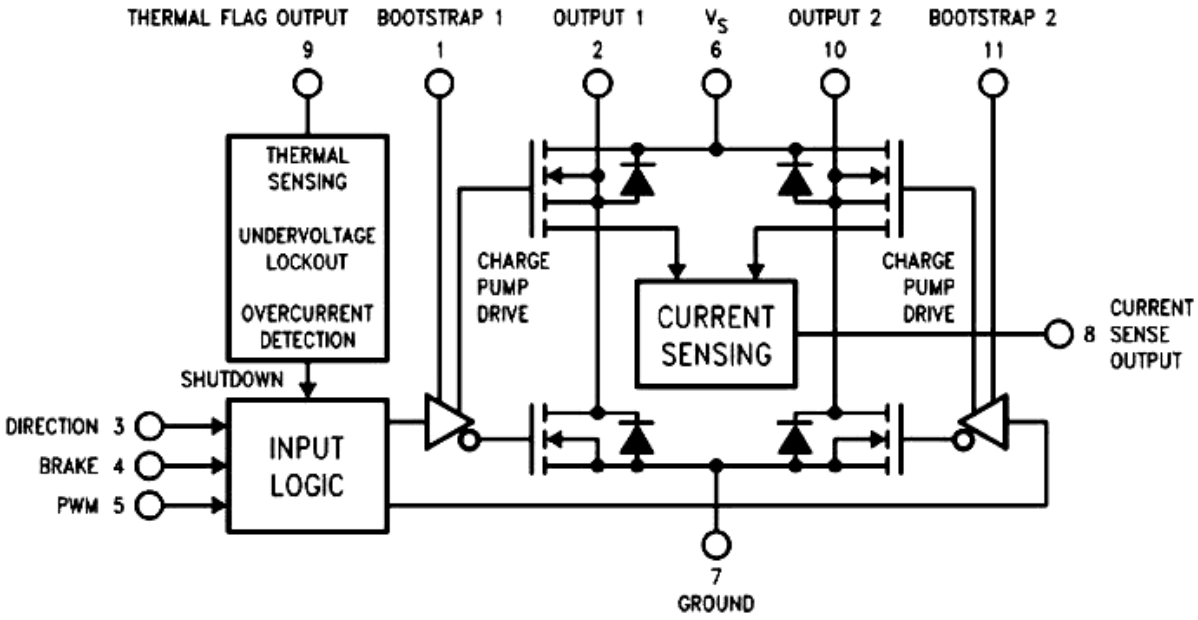


Figure 14. Motor controller function diagram (From [Ref. 8]).

4. Pulse Width Modulator (PWM)

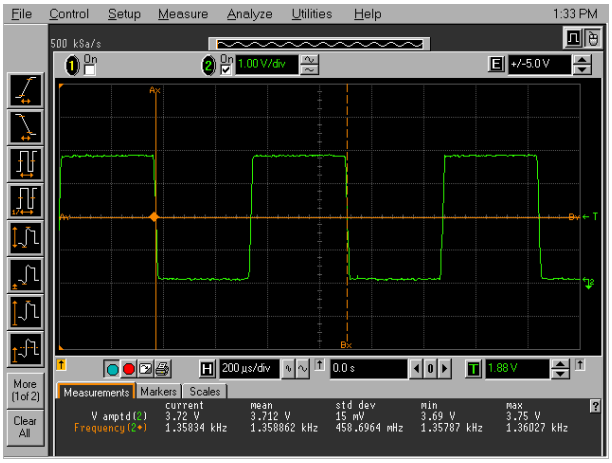
AGV uses a PWM signal from an external circuit that was originally designed to work with Agbot. The circuit was modified for an appropriate frequency for AGV. Figure 16 (A) shows the actual circuit on AGV while (B) shows the

values of the components in the circuit. The circuit begins with a LM555 chip that produces a modified triangular wave. The frequency is given by:

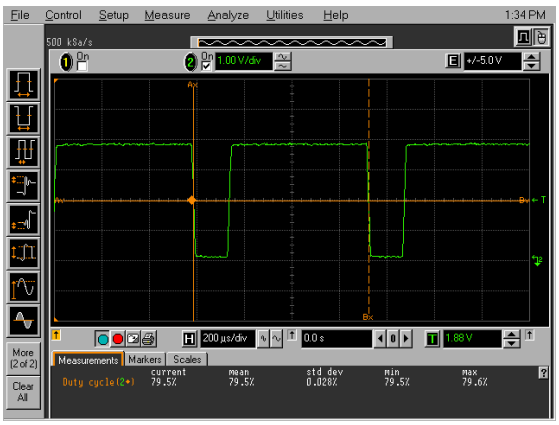
$$F = \frac{\left(R_1 V_{cc}\right) - \left(0.6\left(R_1 + R_2\right)\right)}{\frac{2}{3} V_{cc} R_F \left(R_1 + R_2\right) C_1} \cdot$$

Equation 1. Frequency of the PWM circuit

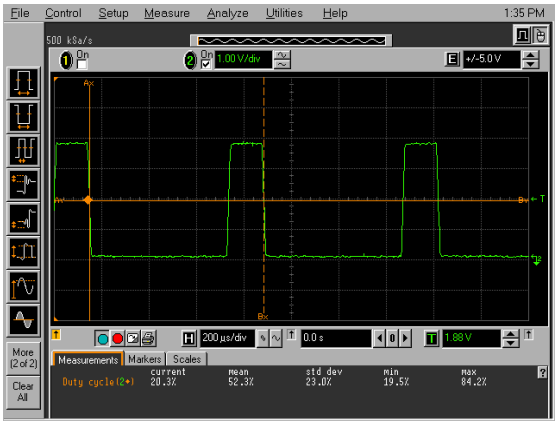
Based on the amount of noise on the output wave, testing multiple frequencies showed that 1.1 KHz was optimal for AGV. The modified triangular wave feeds into a LM324 chip, which is simply multiple operational amplifiers configured as a comparator. The wave goes into the positive input of two separate comparators (one for each motor controller). The negative input of the comparators comes from the BL2000 analog outputs. The desired speed of AGV is computed in the code, which corresponds to a voltage. That voltage is sent through an analog output of the BL2000 (one for each motor controller). The voltage falls between the extremes of the modified triangular waves voltages from the LM555. In this arrangement, the comparator then forms a standard zero to 5-volt square wave. The square waves feed into pin D of the motor controllers and they send the signal to the motors. The duty cycle of the square waves determines the speed of the motors. For AGV, a duty cycle of 0% (straight line 5 volts) represents full speed reverse and a duty cycle of 100% (straight line 0 volts) represents full speed forward. There is a linear relationship for corresponding duty cycles and speeds [Ref. 3]. Figure 15 shows typical waveforms that are sent to the motor controllers on AGV.



(A)



(B)



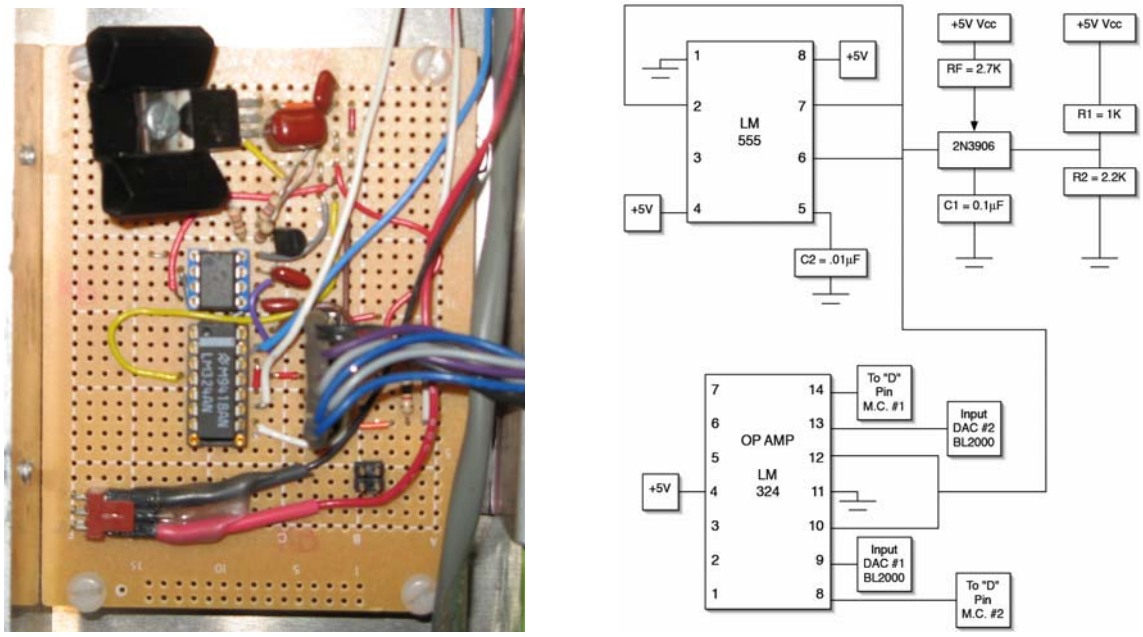
(C)

Figure 15. Typical waveforms sent to the motor controllers. (A) represents the motor controllers telling the motors to stop (an approximate 50% duty cycle). (B) shows an approximate 25% duty cycle that corresponds to half speed reverse. (C) shows an approximate 75% duty cycle that corresponds to half speed forward.

Turning AGV is nothing more then having the BL2000 send a different voltage for each side of the platform that corresponds to a different waveform to the separate motor controllers. Table 2 lists the typical values for the pins on the motor controllers that correspond to the output (speed) by the motors.

Pin P (PWM)	Pin D (Direction)	Pin B (Brake)	Output
H	H	L	Full speed reverse
H	L	L	Full speed forward
X	X	H	Stop
L	X	X	Stop
H	50% DC	L	Stop
H	25% DC	L	1/2 Speed reverse
H	75% DC	L	1/2 Speed forward
H = High (5-volts), L = Low (0-volts), X = H or L, DC = Duty Cycle			

Table 2. Possible combinations of the motor controller pins and the PWM circuit.



(A) (B)

Figure 16. (A) The actual PWM circuit on AGV. (B) The PWM circuit diagram.

B. SENSORS

1. Ultrasonic Range Finder

After AGV receives a command to move autonomously to a new waypoint it goes into navigation mode. In that mode it periodically checks to see if there is an obstacle in its path. There are seven sensors associated with the collision avoidance. The primary forward-looking sensor is

the SRF08 ultra sonic range finder. Figure 17 shows the sensor along with the first three IR rangers mounted on the front of AGV. Figure 18 shows the beam pattern the sonar emits. It can detect and respond to obstacles within this pattern.

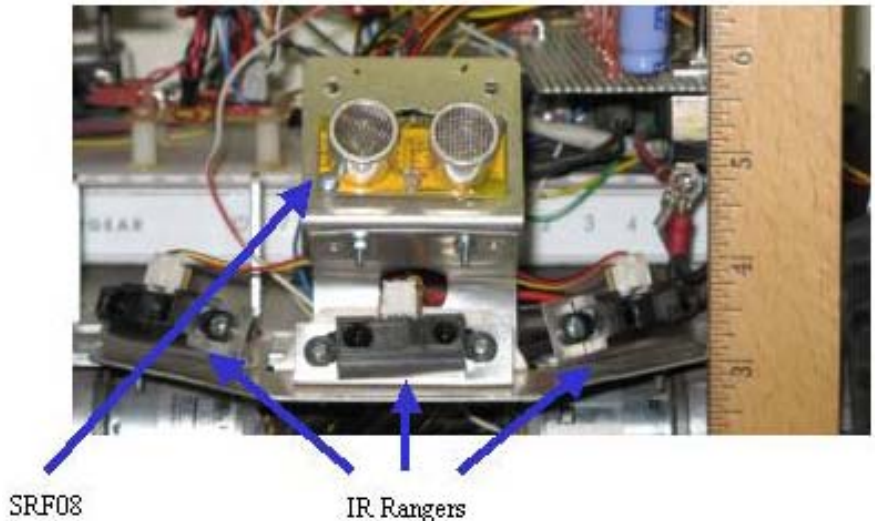


Figure 17. Forward-looking sensors for collision avoidance.

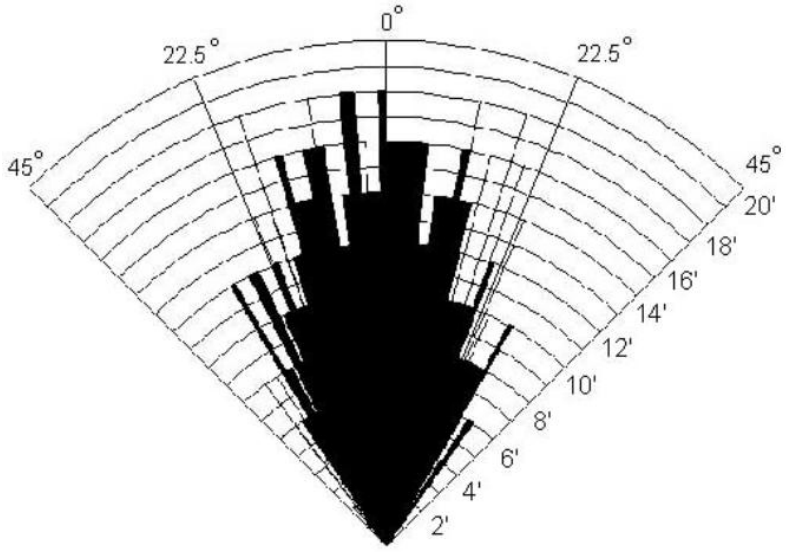


Figure 18. The beam pattern emitted by the sonar (From [Ref. 9]).

The sonar has a maximum range of six meters or 236 inches. The control system allows the BL2000 to read an object at a distance in centimeters or inches. The computer code reports an obstacle at 30 inches and forces AGV into avoidance. The sonar is a basic ultrasonic sensor. The transmitter emits an ultrasonic pulse or energy (at a frequency of 40 kHz) [Ref. 9]. An object within the beam pattern then reflects that energy uniformly within a solid angle (this angle may approach 180 degrees). The frequency of the reflected wave is different than that of the transmitted wave [Ref. 10]. This difference is then converted into a distance using the following formula:

$$L_o = \frac{vt\cos\Theta}{2}$$

Equation 2. Distance to an object from the ultrasonic rangefinder (From [Ref. 10]).

In this equation t is the time the ultrasonic wave takes to hit the obstacle and then return. v is the speed of the wave. Figure 19 shows the typical design for an ultrasonic detection sensor. The angle Θ is the same as the one referenced in Equation 2.

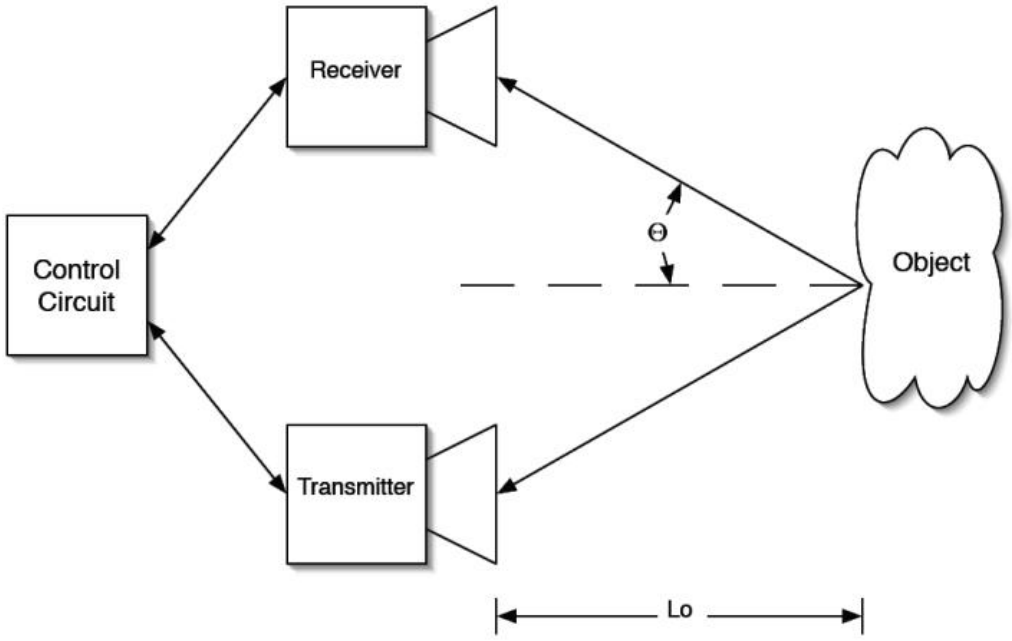


Figure 19. A generic ultrasonic detection sensor (From [Ref. 10]).

To generate the ultrasonic wave the sensor uses a piezoelectric transducer. In the transmitter, a voltage is applied to the piezo ceramic element. This causes the material to flex and emit the wave. Conversely, when the wave returns and hits the receiver it causes a flex in the piezo ceramic element that generates a voltage [Ref. 10]. The sensor is designed for continuous transmission (although when combined with the IIC code this is not exactly what the AGV does). Therefore, it needs both a transmitter and receiver but they are identical in design.

Communication with the ultrasonic detection sensor is through the IIC protocol. Although the ultrasound is the only sensor that requires IIC, the power bus contains additional ports to add other IIC sensors. The ultrasound is also equipped with a front facing light sensor. AGV does not currently use this feature.

2. IR Rangers

AGV has six infrared rangers. Three are mounted in the forward direction to detect obstacles while navigating autonomously (see Figure 17). They cover the same basic area that the ultrasonic range finder covers. However, the IR rangers have proven to be much more reliable than the ultrasonic range finder. In addition to looking forward, two rangers look to each side. Once AGV detects an obstacle (from any of the forward looking sensors) it references each of the side rangers (see Figure 20). The side rangers tell it which side has the most clearance (greatest distance to an object). AGV then chooses to turn away from the obstacle based on the information the side rangers provide it.

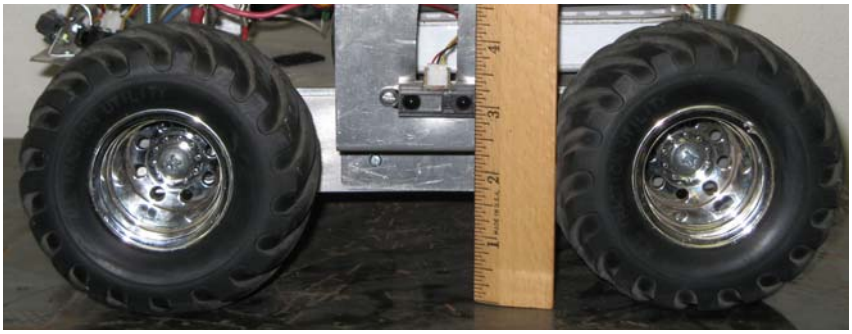


Figure 20. The side facing IR rangers.

Each IR ranger is a GP2D12 from Sharp. In testing, they can accurately measure range from approximately five centimeters up to 80 centimeters. They are analog based with a voltage reported from 0 to approximately 2.6 volts. The rangers use triangulated IR light to detect the distance to an object. The emitter continuously emits IR light at a wavelength of approximately 850 nanometers. If an object reflects that light, then the detector collects

it. With a constant location between the detector and the emitter, the ranger can calculate the distance to the object [Ref. 11]. It generates a voltage that corresponds to that distance. Figure 21 shows the non-linear graph of voltage versus distance. For the forward-looking sensors, AGV has a set voltage (or distance) that it knows it has to maintain in order to not classify an object as an obstacle.

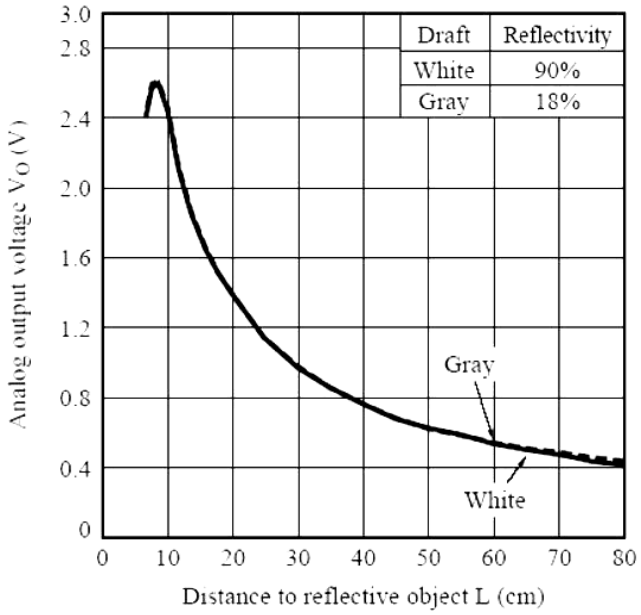
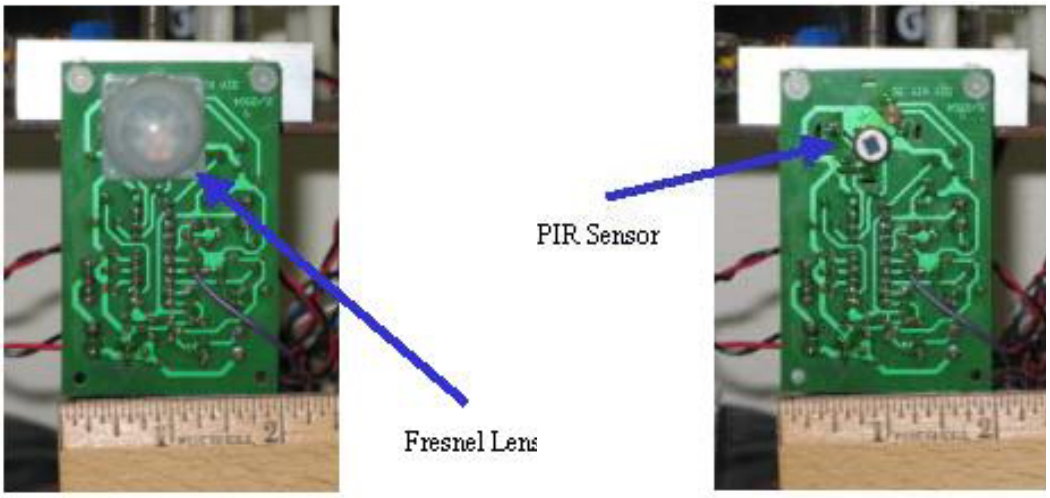


Figure 21. Output voltage versus distance to objects for the IR ranger (From [Ref. 11]).

3. Pyroelectric Infrared Motion (PIR) Detector

To detect a human presence placing an IED, AGV has two types of motion detectors. The first is the PIR motion detector. Figure 22 shows the detector mounted to the front of AGV.



(A) (B)

Figure 22. (A) is the PIR motion detector with the Fresnel lens mounted. (B) shows the PIR sensor and the daylight detection sensor underneath.

PIR motion detectors are preferred by industry over other sensors for their range and ability to not report false detections. This is a sensor that advertises it can report motion up to 25 meters (using a Fresnel lens with a focal length of approximately five centimeters) [Ref. 12]. During testing AGV could reliably report motion to approximately 15 meters. The discrepancy is more than likely due to interference from the router. The sensor has three main components: Fresnel lens, PIR sensor, and a daylight sensor. AGV does not utilize the daylight sensor (when activated it detects motion regardless of the amount of ambient light).

Humans produce infrared radiation that is concentrated within a spectral range between four to 20 micrometers (of course so do many animals making it difficult if not impossible to distinguish). The inside of a typical PIR detector is depicted in Figure 23. The Fresnel lens breaks

up the detection area into optical zones. When a human presence is located within the detection area and the presence moves from one zone to another, it generates a heat wave [Ref. 12]. The heat wave causes the front side of the pyroelectric material, seen in Figure 22 (B), to expand. The stress in the material then causes a piezoelectric charge on the electrodes. The charge is manifested as a voltage on the opposite side of the material. This voltage is then amplified and used as output to report a positive contact. The Fresnel lens in combination with the PIR detector also works in reverse, or if the human presence is cooler than the ambient temperature. In the first case, (the human presence is warmer than ambient) the flux across the detector is positive. If the human presence is cooler than the ambient temperature the flux across the detector is negative, but it will still drive a current and then a corresponding voltage is produced that can be amplified and used as output [Ref. 13].

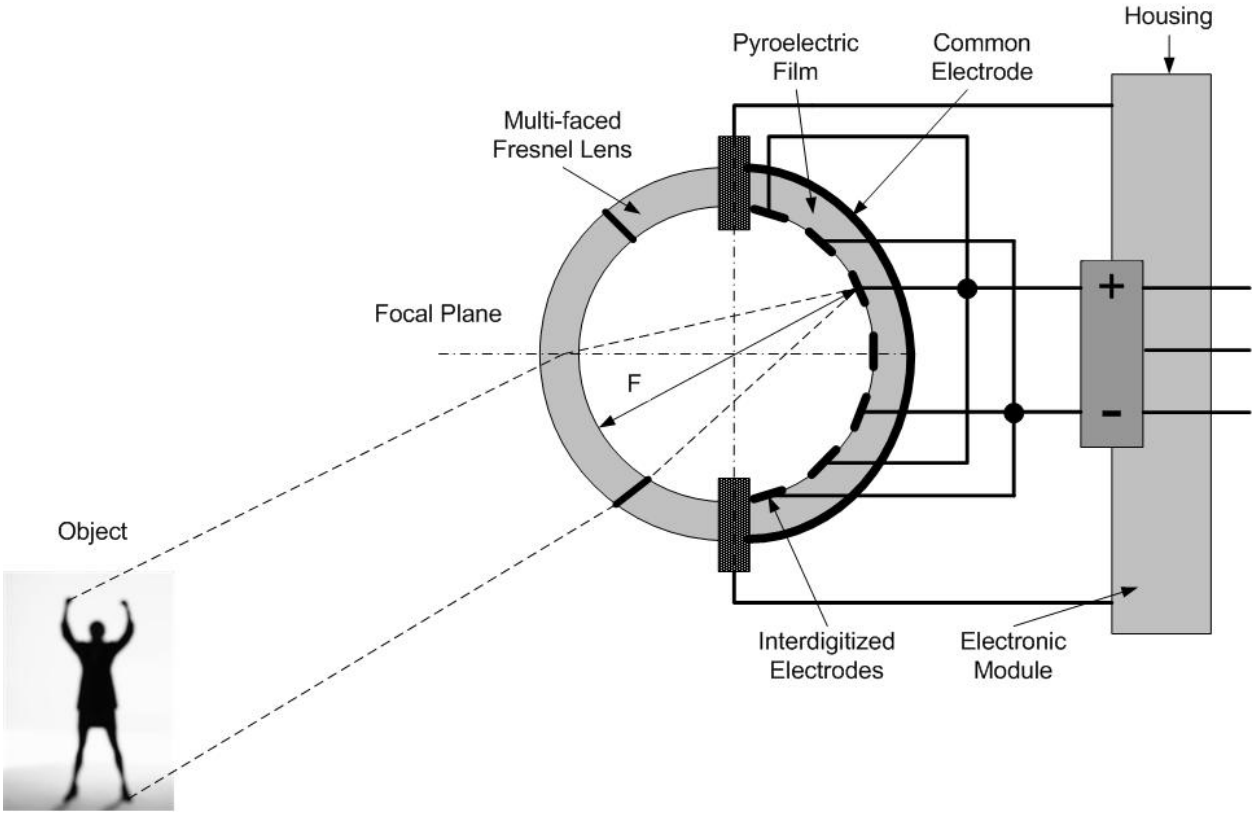


Figure 23. A typical PIR detector with a Fresnel lens attached (From [Ref. 13]).

The PIR detector is designed to drive a device (siren or light) when it detects motion. AGV does not currently use this feature. In order to simplify the output as read by the BL2000, AGV has a hard wire connection from the light emitting diode (LED) on the PIR board. Usually the LED is used to simply light up when motion is detected. AGV uses the voltage that the detector produces for the LED and reports that as a positive contact. When the motion no longer exists, the LED takes a few seconds to settle back to a non-lighted state. This is filtered out in the actual computer code for reporting a detection.

4. Ultrasonic Motion Detector

AGV is also equipped with a second, ultrasonic motion detector. Figure 24 shows the detector mounted facing

outward from the side of AGV. The ultrasonic motion detector operates on the same principles as the ultrasonic range finder. It has a transmitter and a receiver. The transmitter continuously sends out an ultrasonic wave at 40 kHz (6 millimeter wave length). The receiver then picks up any reflected wave and the circuitry amplifies it for the first time. The detector constructs an envelope for the first signal received at the 40 kHz. When there is no movement the envelope is simply a straight-line voltage. The circuit detects movement by recording anytime the signal goes outside of the envelope (the receiver picks up the wave that was out of phase with the original one) [Ref. 14]. In testing, this sensor picked up movement at approximately 10 meters. This circuit is designed to output a voltage of approximately 1.5 volts whenever it detects motion and zero volts when it does not. This output signal is directly fed into the BL2000. This sensor is greatly affected by the router when the router is transmitting information back to the user. AGV uses computer code to filter out these false contacts. Additionally the mounting position (elevated and forward of the router antennas) helped to alleviate the problem.

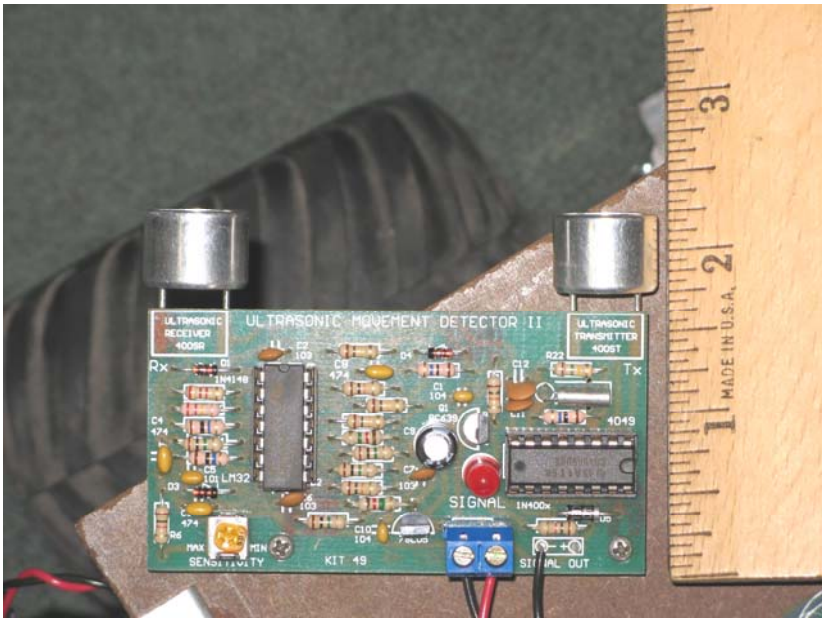


Figure 24. Side mounted ultrasonic motion detector.

C. COMMUNICATION

AGV communicates via a standard 802.11GHz wireless router. It uses the Netgear 240 Mbps, ultra fast Range Max wireless router. The BL2000 has an ethernet cable connection and the dynamic C code has periodic functions that it sends back to the JAVA GUI when interfaced. The connection with the computer is a standard TCP/IP interface. The router has three extra ethernet cable connections that can be used for an external camera or any number additional components. Figure 25 shows the router that AGV employs.

Figure 25. AGV's communication platform.

D. BL2000

AGV's computer processor is the BL2000 Wildcat from Z-World. The BL2000 has a single-board computer with a Rabbit 2000 microprocessor operating at 22.1 MHz. There are 11 analog inputs and 2 analog outputs. Additionally the BL2000 contains four serial ports [Ref. 15]. Figure 26 shows the BL2000 employed by AGV. The BL2000 retrieves input instructions from the user via the Java GUI and conversely sends information back to the user that is displayed on the GUI. In autonomous mode the BL2000 receives input (in the form of analog voltages) from the ultrasonic range finder and the IR rangers for collision avoidance. Additionally, in detection mode it receives data from the motion detectors in order to report a human presence detection back to the user. The BL2000 incorporates all of the input and dynamic C code (discussed later) stored on the flash-ROM then maneuvers AGV or reports contact back to the user.

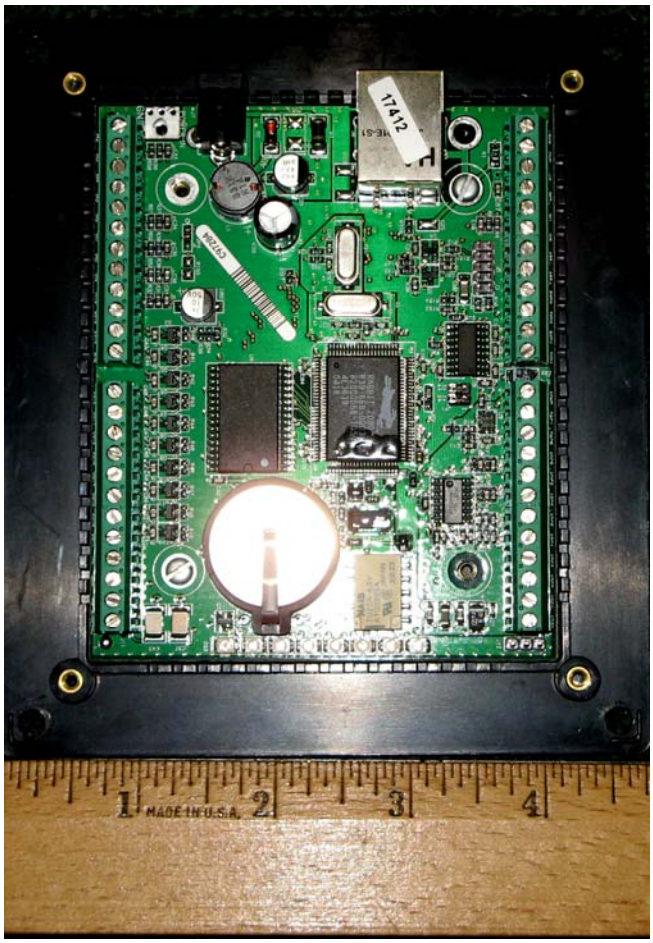


Figure 26. BL2000 rabbit microprocessor.

E. COMPASS

AGV employs a HMR3000 digital magnetic compass. The compass has three magnetoresistive magnetic sensors for determining a heading to within +/- 0.5 degrees. The compass is equipped with a liquid filled two-axis tilt sensor to provide tilt and roll data of up to +/- 40 degrees. Once calibrated, the compass compensates for distortion due to ferrous objects and stray fields. The compass sends an ASCII output via the BL2000 for display on the JAVA GUI [Ref. 16]. The tilt and roll feature of the compass is displayed on the GUI, however the data are not

currently used by the BL2000 to affect any change in AGV's behavior. Figure 27 shows the compass.

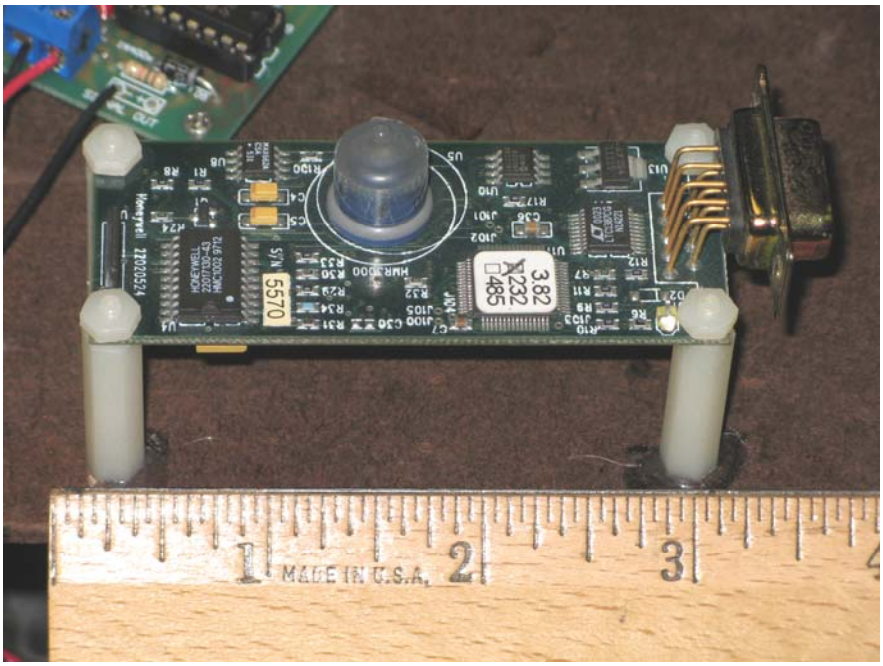


Figure 27. Magnetic HMR 3000 digital compass.

F. GLOBAL POSITIONING SYSTEM (GPS)

AGV uses the Garmin GPS 16 LVS (Low Voltage System) exclusively for navigating (there is no back up dead reckoning system although one could be created). Figure 28 is the GPS employed by AGV. The GPS sends AGV's location through the BL2000 back to the GUI. The location is in standard latitude and longitude coordinates [Ref. 18]. Also included in its transmission is the number of satellites currently being tracked. When navigating, AGV takes the current location and then determines the heading it needs to travel on to get to the new location (discussed later) [Ref. 3].

The GPS has the capability to utilize the WAAS (Wide Area Augmentation System) while in North America. WAAS was

originally a system to improve the GPS position for aviation, but transitioned to also improve land based applications (reference spec sheet). Currently there are two WAAS satellites in operation (one over the Atlantic and one over the Pacific Oceans) [Ref. 17]. The two satellites work in conjunction with the 25 ground stations to calculate possible GPS errors (clock drift, orbital errors, atmospheric delays, etc.). The satellites broadcast the errors and the GPS receiver then compensates for them [Ref. 3]. In testing, AGV utilizes the WAAS (showed by a "differential fix" in the GPS section of the GUI) about fifty percent of the time.

Figure 28. GPS 16 LVS.

G. CAMERA

The detection portion of AGV employs the D-Link DCS-900 internet camera. The camera is cylindrical with dimensions of 2 ½ " x 2 ½" x 2 ¾" and weighs 0.61 pounds.

It is a standard internet camera that plugs directly into the router with a stand-alone IP address. The array size is 640x480, 320x240. It has an auto frame rate along with automatic brightness and contrast control. The focus is manual but in testing a standard setting was adequate. The lens has a 6.0 mm focal length. The camera is limited to daylight only (future versions of AGV will incorporate either thermal or IR images). One of the biggest drawbacks of the camera is its 5-volt / 2.5A requirement [Ref. 19]. Only the router requires an equivalent amount of current. Figure 29 shows the camera. Currently the camera is mounted on AGV's shelf, above the motion detectors (future locations will include a more protected area).

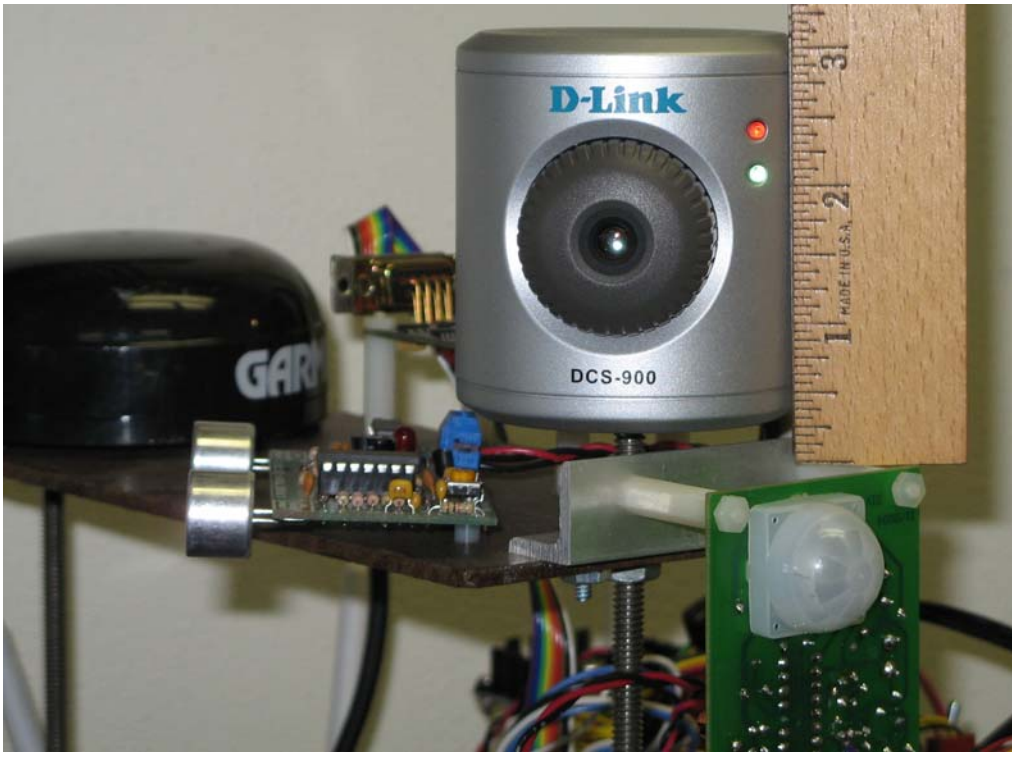


Figure 29. D-Link, DCS-900 camera mounted on AGV.

H. JAVA GUI (GRAPHICAL USER INTERFACE)

The Java GUI was originally written by Kubilay Uzun, an NPS Student [Ref. 21]. AGV employs the same basic program with minor modifications for different driving voltages and different text messages reported from additional sensors. The interface allows the user to input up to 10 waypoints for AGV to navigate to (with the option of stopping or turning at each point). It also allows the user to control AGV in manual mode. The joystick option is extremely useful while investigating a particular area of interest. The upper left portion of the interface (see Figure 30) displays all of the current compass and GPS data. The lower left portion of the interface displays all of the current functions AGV is performing. It also displays the detection statements while AGV is in detection mode. Any standard graphical image can be displayed in the map (center) section of the interface. In many tests a simple “Google Earth” map was incorporated into the interface. The program allows the user to scale the map to the appropriate latitude and longitude locations. Updates were made to the program to include a button to take a snap shot photo with the camera and to incorporate positive contact text messages from the motion detectors.

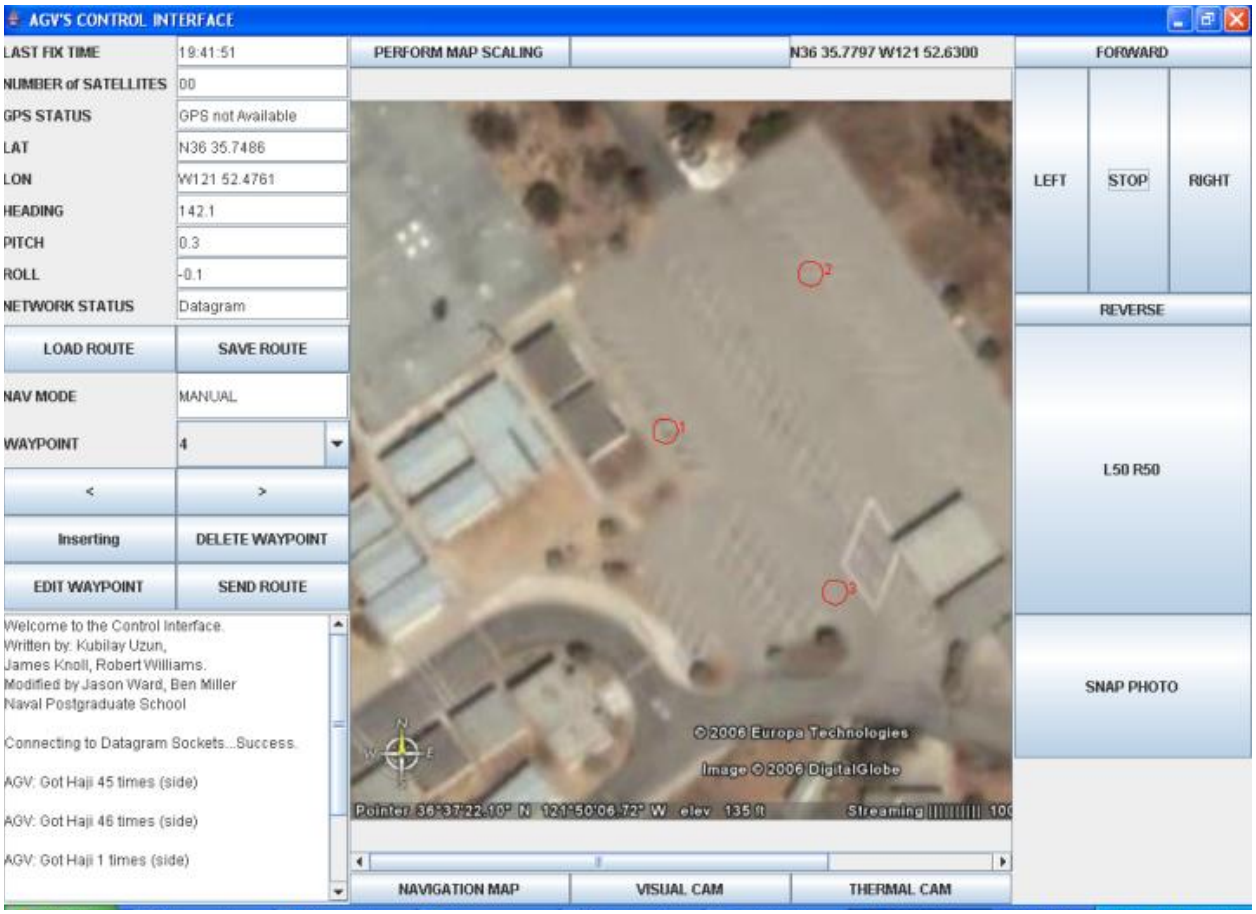


Figure 30. Java Graphical User Interface (GUI).

IV. AGV PROGRAM

The complete Dynamic C code is contained in the Appendix. Figure 31 shows an outline of the program hierarchy.

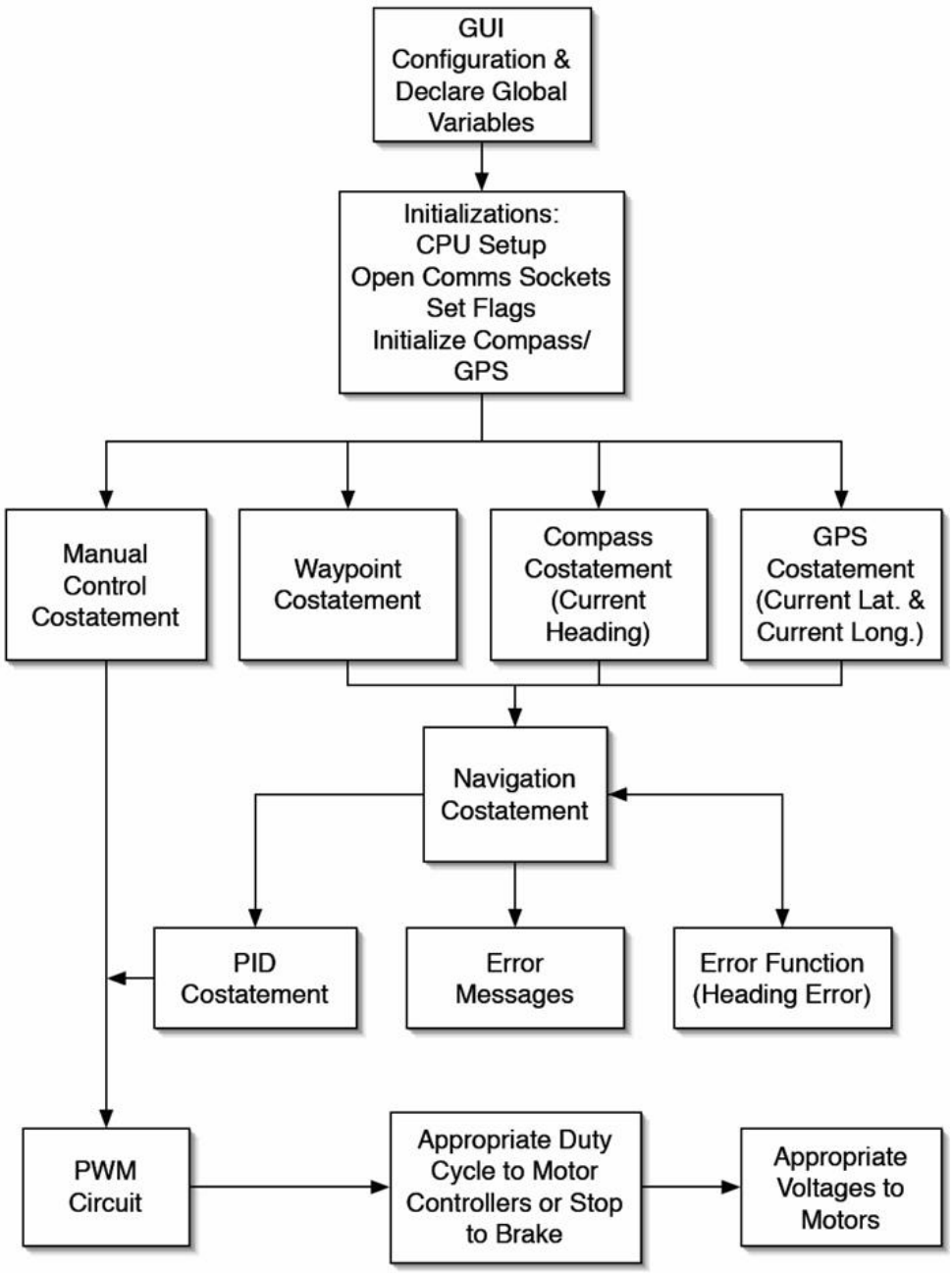


Figure 31. Dynamic C program hierarchy with costatements included (Modified from [Ref. 3]).

A. MANUAL CONTROL

After the initialization functions for the IIC protocols and the communications setup, AGV's program begins by determining if the user wants to put it in manual mode. In manual mode, all other functions and costatements excluding updating the GPS position and compass heading are not utilized. AGV's BL2000 receives a signal from the user via the GUI (reference Figure 30). If either the joystick mode or the driving buttons are employed then AGV is completely controlled by the user. The Java program computes the voltages that AGV's PWM circuit needs for the desired direction. The BL2000 then outputs them to the PWM circuit (which drives the motor controllers and then the motors, reference section III. A. 4.). The original version of the program had Bender stopped with a 50% duty cycle. AGV will stop with approximately the same duty cycle, but each motor controller requires slightly different duty cycles to get its particular side to completely stop (the right side requires 2.45 volts from the BL2000 and the left side requires 2.40 volts). Additionally, the motors have a slight but audible whine noise when they are stopped in this fashion. The problem was alleviated by having the BL2000 turn on the brake (or make pin B on the motor controllers go high) when the user sent a voltage that was within a small window that would have virtually stopped AGV anyway due to surface friction.

B. GPS & COMPASS

The compass and GPS are updated continuously regardless if AGV is in manual mode or operating autonomously. The compass costatement receives the serial

data from the compass itself and converts it into a simple 360-degree heading. The data are then tokenized and sent back to the user. There are possible error messages generated by the code. For example, if the pitch and roll data are outside of the limitations for the compass to detect, an error message is generated. Additionally, if the heading data is corrupt, an error message is also generated and sent to the user. The BL2000 will still continue to try and receive appropriate data, but AGV will not be able to navigate autonomously with a compass error.

The GPS costatement works the same as the compass. The data the GPS sends include the latitude and longitude positions in one line. The program breaks up the line and converts it into an updated location that is sent to the user. If the GPS is not functioning or not updating for whatever reason, the program will continue to report the last known location for a short time. The code then recognizes the data are not accurate. It reports the GPS is not available to the user, and automatically places AGV in manual mode for the user to maneuver (and turns on the detection mode).

C. WAYPOINT NAVIGATION

One of the best features of AGV is its ability to navigate to up to ten waypoints autonomously. The user puts in the waypoints and sends them to AGV from the GUI. The waypoint data costatement then takes each of the waypoints, tokenizes the strings they came in and puts them in the correct order. Once the code determines that it has received all useable waypoints, it takes AGV out of manual mode (and turns off the detection mode if it was

activated), sends a message to the user that AGV is now in autonomous mode, and sends the data to the navigation costatement.

The navigation costatement is the heart of AGV's autonomous operation. The code first determines that the user did not want AGV in manual mode after inputting the waypoints. Simply sending waypoints to the BL2000 does not automatically mean AGV goes into autonomous mode. If not in manual mode, the code begins by converting the latitude and longitude for the current location and the waypoints into decimal values in minutes. This simplifies future calculations for the desired heading. The code then computes the range to the first waypoint (actually in yards, but it could be converted to any units). If the first waypoint does not fall within the range error the program then calculates the new heading that AGV needs to travel. Once the new heading and range are determined, the code determines the heading error from its current heading and the new desired heading. Currently the heading error allows for a 5-degree fluctuation. For example, if the new heading is 250-degrees and the current heading is 247-degrees, then the reported heading error comes back as zero. The range error, set at three yards, operates on the same principle. Therefore, if the range to the waypoint is less than three yards the program designates that as close enough and loads the next waypoint. The navigation costatement is called throughout movement, but the actual voltages sent to the PWM circuit come from the PID statement.

The navigation statement has additional features that aid the user. For example, once a waypoint is passed the program sends a message back to the GUI indicating that the waypoint has been cleared and AGV is proceeding to the next one. The costatement has built in error reports that are sent to the user if problems arise with the GPS or compass data. As standard procedure, AGV goes into manual mode and turns on the detection costatement once any error occurs. Additionally, the autonomous navigation can be interrupted at any time by the user and placed in manual mode.

D. PROPORTIONAL INTEGRAL DERIVATIVE (PID) CONTROL

AGV originally had four speeds that were just a percentage of the maximum speed for turning towards the appropriate heading. The closer AGV got to the heading, the slower the wheels turned. Additionally, AGV originally turned toward the heading like a pivot steering tank (one side of the wheels in reverse with the other side turning forward). This proved to be problematic. AGV has a maximum speed of approximately seven miles an hour, which is much greater than any of the previous robots in the SMART program. AGV would begin to turn towards the correct heading, but would then greatly overshoot it. The BL2000 could not send the lower percentage speed fast enough to get the wheels to slow down in time. Increasing the number of speeds or starting at a lower percentage was ineffective in controlling the turn.

AGV now uses PID exclusively to control its autonomous movement. AGV first maps the heading error into an equivalent voltage to send to the PWM circuit. As an approximation, 3.5 volts to the PWM circuit is full speed forward. 2.42 volts stops AGV, and 1.0 volts puts AGV into

full speed reverse. These voltages are approximate values independent of which side the motor controllers are controlling. When AGV's wheels are not in contact with any surface, one can see that these voltages are far from exact. In developing the mapping equation, a simple linear relationship was used. Mapping the full speed forward voltage to the absolute value of the heading error and then the stop voltage to a zero heading error gives Figure 32.

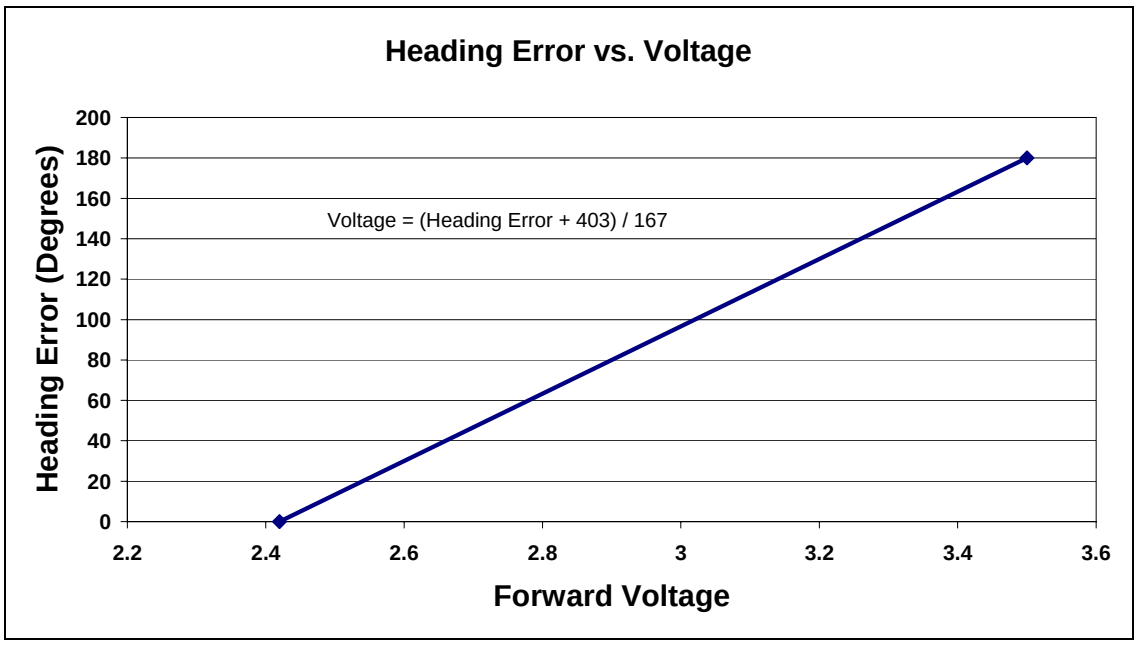


Figure 32. Heading error mapped into the appropriate voltage.

The maximum absolute value heading error can only be 180 degrees (which corresponds to the maximum forward voltage, 3.5 volts). Starting with the simple equation for a straight line:

$$y = mx + b \ .$$

Equation 3. Equation of a straight line.

The slope of the line is given by:

$$m = \frac{180 - 0}{3.50 - 2.42}.$$

Equation 4. Slope of the Heading Error / Voltage line.

The slope approximately equals 167 degrees per volt. The intercept of the line is given by:

$$b = -\frac{(180)(2.42)}{3.5 - 2.42}.$$

Equation 5. Intercept of the Heading Error / Voltage line.

The intercept equals a negative 403 degrees. Therefore, the equation for a voltage given the heading error is simply:

$$Voltage = \frac{Heading\ Error + 403}{167}.$$

Equation 6. Forward voltage given a heading error.

In the first trial AGV had no integral or derivative component of the PID. The code simply multiplied the voltage equation by a gain factor to overcome the inertia of AGV and sent that voltage to the PWM circuit. However, just proportional gain proved to be insufficient (or under damped). When moving towards the right heading, AGV always had an offset. When the gain was increased to overcome the inertia of AGV, the maneuver became chaotic. AGV did not simply overshoot the heading on a consistent basis. The processing time the program took to adjust the heading was slightly different depending what other costatements were active concurrently. This, coupled with the possibility of different surfaces (all with different levels of friction),

caused the chaotic motion. These trail and errors lead to the development of AGV's current control system.

Leaving AGV in an under damped state and then adding an integral component for a gain factor was fairly successful. Each time the program loops through the PID costatement it increases the amount of integral gain that AGV uses by a small amount. This occurs up to 40 cycles. In indoor tests on a constant surface, AGV was able to find the correct heading with minimal oscillations. However, once AGV moved outdoors, the ability to move to the correct heading became inconsistent.

AGV's complete PID costatement combines proportional, integral, and derivative gains. Additionally, through multiple trails on different surfaces, AGV performs much more consistently when the movement towards the correct heading happens while moving forward instead of "turning on a dime" or while stationary. The final equation for the voltage sent to the PWM circuit is given by:

$$Voltage = P\frac{Heading\ Error + 403}{167} + I\left(Scalient\right) + D\frac{Heading\ Error + 403}{167}$$

Equation 7. PWM voltage.

Scalient is a variable in the computer code that is an adjustable percentage of the total voltage sent. The coefficients P, I, and D were experimentally determined to optimize the performance independent of the surface AGV traveled on. After multiple tests on different surfaces the coefficients were determined as follows: P = 1.1, I = (0.05)P, & D = (0.05)P. Additionally, the tests showed that an optimal turn toward to the correct heading occurs

when one side of AGV goes forward from the voltage determined in Equation 7, while the other side of AGV continues to move forward, but with a voltage at 85% of Equation 7. The PID costatement contains checks to ensure that the PWM circuit does not receive voltages that are more than the full speed forward voltage (approximately 3.5 volts). It also contains the same checks to ensure that the wheels never go into reverse while moving towards the appropriate heading.

Determining the most efficient direction AGV should turn to move towards the correct heading is a simple exercise in geometry. When drawn completely out, the exercise culminates with AGV turning left when:

$$Heading\ Error \geq 180^{o} \quad \text{or} \quad 0^{o} < Heading\ Error > -180^{o}.$$

Equation 8. Heading error for a left turn.

For all other cases AGV turns right to achieve the appropriate heading. Allowing the heading error to be greater than 180 degrees or less than –180 degrees is a departure from the previous technique for calculating the correct voltage. This is just bookkeeping task in the computer code.

E. COLLISION AVOIDANCE

AGV utilizes the forward-looking ultrasonic range finder and the forward-looking IR rangers (see Figure 17) for collision avoidance. The collision avoidance costatement is only referenced when AGV is autonomously navigating (not in manual control). The costatement begins by calling the ultrasonic rangefinder to determine what objects are within the navigation path. The ultrasonic range finder communicates with the BL2000 via an IIC

connection. IIC is designed to incorporate a large number or sensors operating on only two lines of communication. Although AGV only currently has one IIC component, it could handle many more. In a simplified form, IIC is nothing more than a communication via a clock and a data transmission and works in the following manner. The BL2000 provides the ultrasonic range finder with a standard square wave as a clock transmission. When the BL2000 wants data from the ultrasonic range finder it has to go through a predetermined series. The BL2000 writes a byte to the ultrasonic range finder (through the clock transmission) telling the range finder it wants data from it. The range finder then sends an acknowledgement back to the BL2000 via the data transmission line. The BL2000 then tells the range finder what type of data it wants from it (in AGV's case that data is distance to objects in inches). That transmission is again acknowledged. The range finder then reads a byte (gets the range in inches) and the code assigns it to a variable that the BL2000 recognizes.

Throughout multiple tests, the ultrasonic range finder had a series of problems. First, when the range finder did not detect any objects in its path (distances greater than 20 feet), it reported a zero value for distance to an object. A simple fix, assigning a value of 255 inches when the range finder returned a zero was implemented. An additional problem that did not occur indoors, but did outdoors was false returns. One in every approximately 20 returns reported a false object. After trying many different solutions, the code now takes three returns from the range finder and averages the three to give an approximate distance to objects. Taking greater than three

returns would be optimal, but the time to loop through those returns proved to be too great to accomplish other missions. Throughout the field tests, the ultrasonic range finder reporting an object at 30 inches proved to be enough time for AGV to react and move out of the way.

AGV also utilizes the three IR rangers (see Figure 17) in front to detect obstacles. As discussed previously, the IR rangers are analog so their communication with the BL2000 is greatly simplified. Each ranger is connected directly to one of the BL2000's analog inputs. At the completion of the ultrasonic range finders connection, the code references each of the IR rangers. Through trial and error, voltages greater than 0.2 volts but less than 2.0 volts allow AGV to turn away from an obstacle before contacting it.

If any of the front four sensors reports an obstacle within the determined threshold the code then references the side IR rangers to determine what direction AGV should turn in order to avoid the obstacle. Whichever side has the most clearance (greatest distance to another obstacle), AGV turns in that direction. The code has AGV momentarily stop (in practice this stop cannot be noticed), do an approximate 45-degree turn in the appropriate direction, and then drive forward at approximately three quarters speed. At this point the collision avoidance statement is complete and AGV moves back into the navigation statement. However, if AGV did not clear the obstacle or if there is now another obstacle in its path it will detect it and start the process over again.

The code allows for the possibility that there may be an error in processing, or an obstacle that AGV simply did

not detect in time to alter its course in order to avoid it. At approximately 2.0 volts and greater (from experiment, not theory) the IR rangers have an obstacle within 10-inches or closer and conversely when the ultrasonic range finder reports objects closer than 10-inches, there is one (obviously). Assuming the rear IR ranger reports clear, the code then has AGV go into full speed reverse for approximately one yard. AGV then stops and again references the side IR rangers and the process is the same as before. Ideally a bumper sensor would also be incorporated with this statement, but AGV does not currently have this feature.

F. DETECTION

The detection mode consists of stopping AGV completely and beginning to take inputs from the motion detectors. AGV can turn on the detection mode in three different ways (reference Figure 8). Once AGV completes the manual commands sent to it, assuming there are no subsequent autonomous commands, the detection mode automatically turns on. Upon completion of its autonomous navigation, AGV pauses in order to prevent the motion detectors from reporting its own movement as contact, and then turns on the detection mode. Additionally, if for any reason there is an error in navigation or if the compass or GPS do not function, AGV stops and turns on the detection mode.

The detection statement begins by turning the brake on to prevent any false contact reports from the motors. The code then has the BL2000 read the voltages from the motion detectors. In testing, both motion detectors had a tendency to report false contacts when mounted close to the router. The current location (reference Figure 7) reduces

the effect but does not eliminate it. Powering the motion detectors from a separate 9-volt battery also eliminated some of the interference. As a hardware consideration, this is somewhat inconvenient, but a fully charged 9-volt battery can continuously power the motion detectors for over a month. In testing, the PIR motion detector is affected less by the router than the ultrasonic motion detector. The code has a single filter for the PIR detector. The PIR detector has to detect motion eight times before it reports a positive contact. For the PIR detector, the motion does not have to be continuous.

The ultrasonic motion detector is more prone to detect false contacts. It has the same filter as the PIR detector but the motion for the ultrasonic detector must be continuous for eight loops through the detection costatement. The router has a tendency to cause the ultrasonic motion detector to randomly put out a voltage (report a contact). Testing showed that the two combined filters eliminated the majority of the false contact reports.

The detection statement keeps track of how many times the motion detectors report a contact. When either of the detectors has a positive contact the code generates a text message that includes the total number of contacts that both of the detectors have detected. This message is sent back the user and displayed in the lower left portion of the GUI (see Figure 30). The user then has the choice of simply taking a snap shot photo of what AGV sees or turning to the streaming video portion of the GUI. The snap shots are stored in a folder on the desktop of the user’s computer. The streaming video has a delay of approximately

two seconds. This delay makes driving AGV at a rapid rate difficult, but a slow traverse of the suspected enemy area is possible. Ideally, AGV would leave the camera off until the motion detectors reported a possible contact, but this feature is not currently available. Additionally, testing has shown that a button on the GUI that would clear the number of positive contacts the motion detectors accumulated would be a useful feature. Currently, the only way to zero out the number of positive contacts is to click the stop button in the manual control section or to reset the BL2000.

Additional motion detectors are needed to cover all directions a contact could come from. AGV can turn towards the likely avenue of approach for IED emplacement, but ideally AGV would detect motion in all directions. Currently, AGV covers approximately 180 degrees (forward facing and to the right side).

V. RESULTS

Partial results for the separate components of AGV were included in the above chapters. However, some general analysis can be done by section. The exterior of AGV is not currently suitable for the harsh dessert environment (or any outdoor environment). Ideally, AGV will be contracted out to an engineering firm who will be able to reduce the actual size to approximately a brick. AGV needs to look like its environment. In the Iraqi theatre this means modeling basic trash on the side of the road.

The wheels of AGV are rugged enough for most environments but the packing foam they are stuffed with could be improved. No matter how tightly the packing is, a more uniform material would be optimal. Possibly a liquid gel or even a hardened inner tube could be used. This is most noticeable when AGV attempts to navigate on dirt or loose gravel.

AGV's battery was adequate for an initial prototype, but will not be suitable for the end state platform. In the initial tests, 2000mAhr was sufficient for two hours of on station time. However, once the GUI interface and the router were incorporated onto the platform, the on station time was reduced to approximately one hour (depending on how many waypoints AGV had to navigate to). Additionally, once the camera was added the on station time was reduced even further to 30 minutes. A lithium battery is probably the best solution. However, in the short term an external digital switch written in the code to turn the camera on and off would suffice. The camera could come on only when the motion detectors reported a possible contact and then

turn off after a predetermined amount of time, or after the detectors did not detect any further presence. In the same principle, switches to the collision avoidance sensors could turn them off when AGV was stopped or in manual control.

AGV's autonomous navigation is the heart of its capabilities. In AGV's final test, the PID statement in the computer code works well when the battery is fully charged and the surface is known prior to entering an environment. However, when the surface is drastically different than the norm (i.e. moving from concrete to moist soil) then AGV's performance is greatly reduced. Additionally, when the battery begins to drop past a ¼ charge it affects the motors speed regardless of what speed the BL2000 tells the motor controllers to move at. There is no easy fix for this problem. A possible solution is to develop a program that can detect the resistance the wheels are under and then adjust the PID coefficients accordingly. Another solution may be to research for a compass that responds on the microsecond scale to give AGV adjustments in direction that are not delayed and affected by the terrain it has just traversed.

AGV does not have a back up dead reckoning system to move autonomously to a waypoint. In testing this was only a problem around large trees and tall buildings. AGV would very rarely lose the GPS signal altogether, but frequently would lose its differential fix. Although this is not critical for its operation, it does mean AGV's position was only within +/-10 meters at best. If AGV's mission is to detect a presence in a general area, this position error will not greatly affect it. However, if AGV is assigned to

a very specific location then the motion detectors may not reach out to the extent that the user needs them to. Additionally, when AGV was navigating autonomously with a fix that kept changing from differential to standard, the route it took to the waypoint would appear to be chaotic until the GPS was consistent. If the GPS was inconsistent for any length of time (over 3 minutes) then there is a lot of wasted movement when AGV attempts to stop at its final destination. Increasing the proximity error could help with this, but without significant advances in the GPS technology there is not an easy fix for this drawback.

In the final test, AGV's detection system was simply adequate. The ultrasonic motion detector is significantly affected by the components on AGV itself, the router and the camera most noticeably. Filtering out the false contacts is more of an art than a physics problem. Once the proper filter works for just the onboard components, then the environment has to be considered. Acoustic waves from any number of sources greatly affect the sensors performance. There is a medium that can be reached so the sensor does not over report, and does not miss obvious motion contacts. Unfortunately, for the ultrasonic sensor that medium only occurs when the contact is within a few meters of AGV. The PIR motion detector is not affected by environmental noise and its filter can be adjusted to filter out almost everything but a thermal motion contact. In the future, moving to all PIR detectors should alleviate the false contacts or in not reporting an actual contact.

The camera being continuously on was discussed earlier; however, there are other drawbacks. The field of view is adequate for almost all situations, but the tilt of

the camera does not always correspond in optimally assessing a contact. For example, in the final test when AGV detected a contact and the contact was close to the ground (i.e. digging a hole to place an IED) then the system worked well. But, when the contact was simply moving in the area then the height of the person was too great to capture the face. The positive is the contact is detected and a partial picture is obtained, but if the user desires to capture the complete description of the suspect the camera has to be tilted up. A possible solution is the mount the camera on a pan and tilt servo (separate from moving the entire platform). This solution is feasible, but adds additional processing time for the BL2000 and will add additional power requirements.

Obstacle avoidance during autonomous navigation is essential for this platform. Currently the system is insufficient and needs improvement. Despite carefully defining the field of view for the forward facing sensors and ensuring there were overlap between them, some obstacles were reported before AGV could take action to avoid them. Making AGV more rugged and incorporating a bump sensor will help minimize the effects of hitting an obstacle, but improvements in the basic system can still be made. Additionally, far too often AGV detects an obstacle when none exists. This causes the platform to continually have to correct its current heading and can make the autonomous navigation to waypoints look chaotic. The ultrasonic range finder detects obstacles at almost double the range of the IR rangers. However, the final test showed the ultrasonic range finder is susceptible to the same environmental noise that the ultrasonic motion

detector is. Averaging the returns from the sensor did alleviate some of the false returns, but it reduced the range of the sensor to just beyond the range of IR rangers. Simple solutions may be to increase the number of IR rangers in the front spread or to research for another IR ranger that has a greater range. Another solution may be to put a filter, either hardware or in the code, on the ultrasonic range finder to eliminate the false returns. Figure 33 details how the IR ranger matched its published data. Figures 34 and 35 show how accurately the ultrasonic range finder reported distance to different objects both indoors and out. The figures also show how inaccurate the range finder is when the ultrasonic motion detector is on.

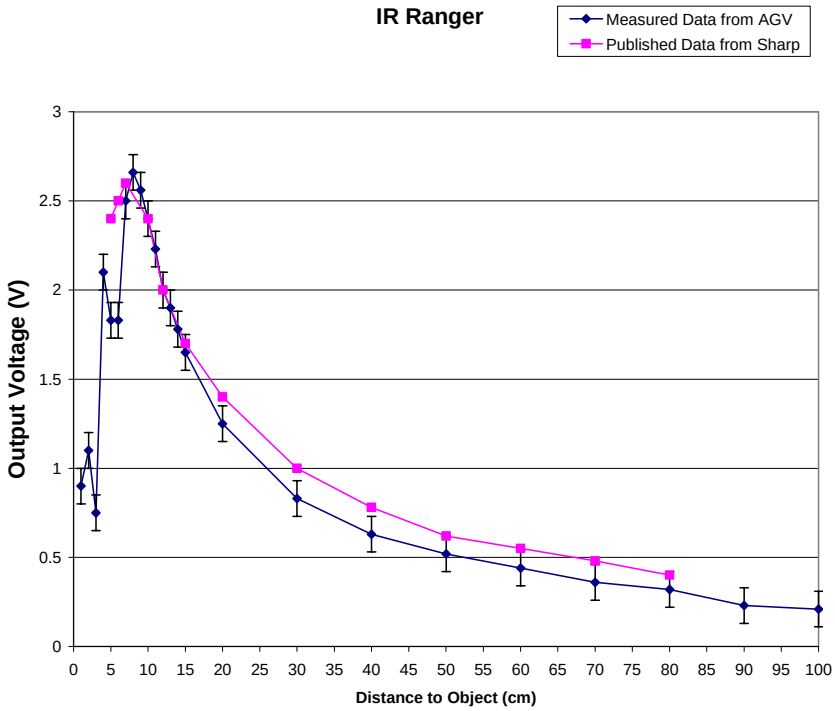


Figure 33. IR ranger output voltage versus the published output voltage from sharp.

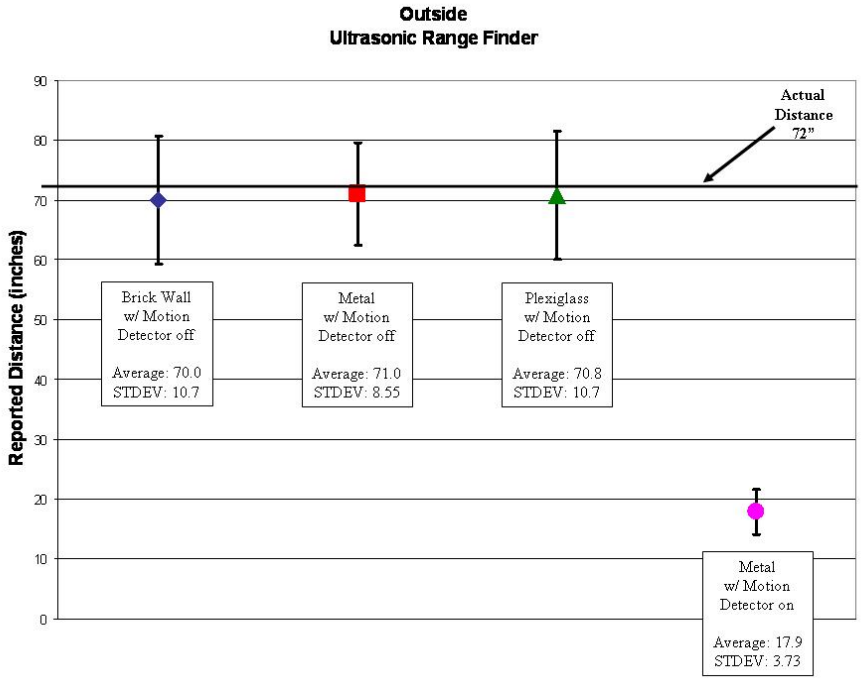


Figure 34. Outdoor object detection by the ultrasonic range finder.

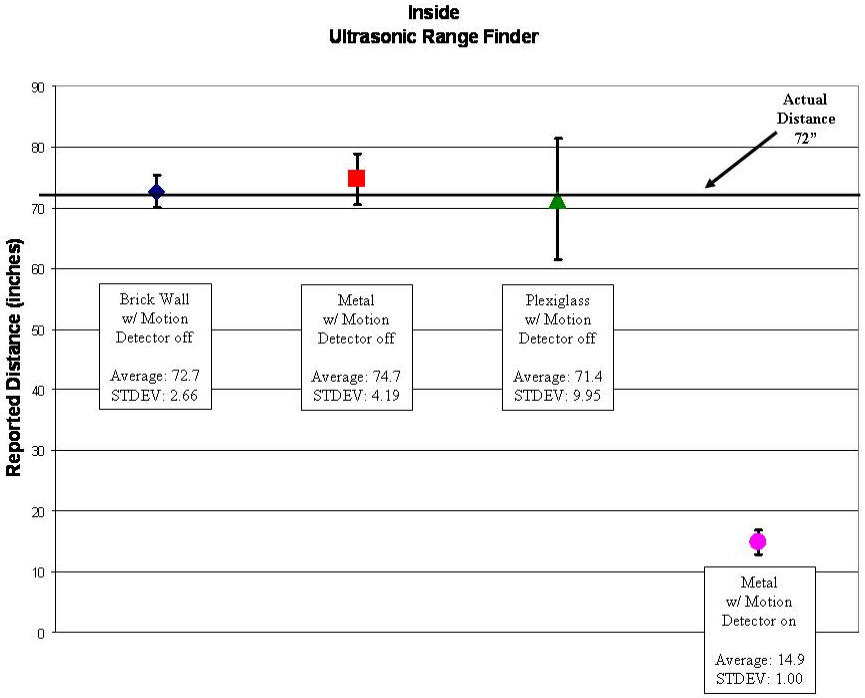


Figure 35. Indoor object detection by the ultrasonic range finder.

The communication platform limits the range AGV can be from the user if the user wants instantaneous feedback. However, AGV will continually transmit its detection data, along with all of the navigational information regardless if the user is within range or not. In the final test, AGV was left in a location while the GUI interface was moved out of range. After moving back within range, the interface picked up the data and displayed the number of contacts that were received while the interface was not connected. Although not ideal, this does show that continuous direct communication is not necessary. There are current research projects that show promise in roving wireless networks able to deploy over large areas. With just the standard 802.11G router, AGV could work well in that environment. In the long term, AGV would move away from the standard router connection and be incorporated into existing communication networks for the Army and Marine Corps (Blue Force Tracker).

THIS PAGE INTENTIONALLY LEFT BLANK

VI. FUTURE WORK & CONCLUSIONS

A. FUTURE WORK

The majority of future work will be integrating AGV into the Blue Force Tracker communications network the Army uses or the Marine Corps equivalent. Introductory research and contacts have shown that this is certainly possible without any major hardware overhauls and virtually no software adjustments. Having each AGV appear as its own icon on the GUI of every friendly vehicle within in range is paramount. With this integration, AGV could not only communicate with the direct user, but all other units and vehicles in the area of operations would know exactly what AGV sees on its current mission.

Integrating a new thermal camera is also a key component of AGV's future development. Although currently ordered for the SMART initiative, basic tests have to be conducted along with meeting additional software requirements. The thermal camera is critical for AGV to perform its mission in the most likely of environments, limited visibility.

The sensor array for the obstacle avoidance is simply inadequate. Future work in either improving the array through hardware and software or in designing new sensors altogether will be essential. Detecting obstacles for avoidance when in fact none exist is not an issue if it only happens periodically. However, when it occurs every few seconds, then AGV does not have the processing power to continue its mission.

The BL2000 is an excellent on board computer for a prototype, but may not be ideal for the end state platform.

The total processing time of the program is on the seconds' time scale. Utilizing Object Oriented Programmable Integrated Circuits (OOPic) would be an excellent way to take some of the basic functions away from the on board computer, freeing up valuable processing time. This could bring the time down to the microseconds' scale. Additionally, OOPics can greatly reduce the power consumption, giving AGV longer time on station. The SMART initiative is researching the OOPic for use on all of their platforms.

The next generation of AGV will incorporate the majority of changes discussed previously. However, the platform itself needs to be hardened and more than likely about twice the current size. No matter how sophisticated the obstacle avoidance is, there are certain surfaces that AGV must always be able to traverse. A larger and more rugged wheel design will be key in allowing AGV to move through light mud to loose gravel without a large adjustment to the PID coefficients. Larger wheels will also increase the ground clearance. This is important for movement over obstacles the avoidance system does not detect.

B. CONCLUSIONS

Overall, AGV accomplishes the mission the SMART initiative set for it. Its current limitations are due in a large part to the hardware engineering inability by the prototype designer and the communications platform. However, even in the current form AGV can: autonomously navigate to waypoints, avoid obstacles, investigate possible threats and then detect motion that triggers a visual camera. The information is then relayed back to the

user and displayed on a fairly sophisticated interface. Hardware adjustments and additional software refinements are needed prior to AGV being fielded to the Army or Marine Corps.

There is not one solution in combating the IED problem. Every proposed solution has advantages and drawbacks. AGV's advantages can provide the user with a small, semi-autonomous, relatively inexpensive platform that can aid in over-watching the main supply routes where IEDs are typically placed. Additionally, AGV can assist in over-watching a specific target for future action. AGV's drawbacks include limited on station time, limited field of view, and a limited communication range. The SMART initiative believes that AGV can quickly be engineered for rapid fielding to combat units where the IED is an everyday reality.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX DYNAMIC C CODE

```
/*********************************************************************
*********
AGV_NAV_ver.c
    AGV Nav is the navigation interface with the BL2000 processor used
to
        control the Physics Department robot known as AGV (Autonomous
Ground
        Vehicle).  Currently, the program compiles with five warnings
        (Dynamic C 7.04P3).
*********************************************************************
*********
Version History:

    --- Version 1 ---
    July - September, 2004
            CDR Jerry Stokes
            LT Sean Niles
            LT Irv Pollard
            LT Jason Ward
      LT Brett Williams

    Changes:
    This version implements the Sonar costatment, allowing Bender to
react to obstacles in his path.  Algorithms to handle impediments to
Bender's planned path have also been implemented.  Nav - pared down to
handle only calculations of GPS positions and generating headings and
ranges for following waypoint paths.  GPS is now initialized to only
give 2 sentences (one cannot be turned off) so that data culling when
parsing the GPS sentence is reduced.  WayPoint - algorithm was
optimized and altered to allow the user to input an exact path desired.
Previous versions used the closest waypoint as the next waypoint to
which to drive.  Control - PID control was implemented as its own
costatement, partly to alleviate the delay associated with taking
manual control when Bender was performing calculations in Nav.
Networking - Ports were shifted to 4001 and higher to avoid conflicts
with reserved ports.  UDP is still implemented, however with greater
throughput realized from 802.11g wireless hub, it may be advantageous
to switch to TCP/IP in later versions.  IP addresses were shifted to
192.168.0 domain for the wireless router.  Should future users decide
to move Bender to the university network, static IP addresses will need
to be obtained for the camera, Bender, and router.

      --- Version 2 ---
      Created for SE4015, Summer 2003
      James Knoll
      Kubilay Uzun
      Robert Williams

      This program was written to run on the BL2000 and control the
Nav, Sensors, and Motors of Bender.  Compiles with two warning in
Dynamic C 7.04P3.  Newer versions of Dynamic C will require
modification in the networking since UDP has changed.

      --- Version 3 ---
      Created for the AGV, Summer and Fall 2006
      Ben Miller
```

```
      This program converts the old Bender code to work with a wheeled
robot.  The communication is now TCP/IP.  The PID is completely
different along with the collision avoidance (there is a portion with
IIC to control the ultrasonic sensor).  The navigation statement was
modified slightly (all the voltages are now sent from the PID
costatement).  There is a new detection costatement that did not
previously exist.
*******************************************************************************
CONNECTIONS

      Nav   to   Motor
      dac1 <--->       //left side wheels
      dac0 <--->       //right side wheels

      Nav   to   GPS
      tx2          RED
      rx2          GRN
      grnd         BLK

      Nav   to   Compass
      tx1          grn
      rx1          red
      grnd         BLK


*******************************************************************************
/

//IIC Settings

#define READDELAY 15
#define MAX_SENTENCE 100


//Network Connections

#define MY_IP_ADDRESS            "192.168.1.81"
#define INTERFACE_ADDRESS  "192.168.1.80"
#define MY_NETMASK         "255.255.255.0"
#define MY_GATEWAY         "192.168.1.1"

#define WP_PORT 4002
#define MAN_PORT 4001
#define COMPASS_PORT 4004
#define GPS_PORT 4003
#define ERROR_PORT 4005
#use "dcrtcp.lib"
#memmap xmem

//Serial Port Settings

#define BINBUFSIZE 127
#define BOUTBUFSIZE 127
#define CINBUFSIZE 127
#define COUTBUFSIZE 127

//GPS Variables

      double curr_lat;
      double curr_lon;
```

```
	const int xmit_delay = 100;

	char sentence[MAX_SENTENCE];
	char dir_string[2];

	typedef struct {
		int lat_degrees;
		int lon_degrees;
		double lat_minutes;
		double lon_minutes;
		char lat_direction;
		char lon_direction;
	} GPSPosition;

	GPSPosition current_pos;   // Declare new GPSPosition variable

	const int gps_delay = 0.5; //seconds to delay between gps readings

	int gps_error, gps_error_count;

	const float pi = 3.14159;

	const char GPS_Reset[]="$PGRMI,,,,,,,R\r\n";  //Unit reset
	const char GPS_Sent_Clr[]="$PGRMO,,2\r\n";    //clears all output
	                                               //sentences
	const char GPS_GGA_Enable[]="$PGRMO,GPGGA,1\r\n";//enables the GGA
	                                                 //sentence

	unsigned long gps_wait_time;
	const int gps_timeout = 1;

//New Detection Statement Variables
	char h[30];
	char c[12];
	int detect_flag, motion_flag, a, b, time_flag;
	float motion1_volts, motion2_volts;

//New PID Statement Variables
	char s[12];   //compass inputs if you want to test the PID
	char p[12];
	float turnvolts, fwdturnvolts;
	float scale, scaleint, P, I, D;
	int flag, flagint, compconv, k, sonarplus;

//New Compass Statement Variable
	float curr_hdg;
	char compass_sentence[MAX_SENTENCE];
	int compass_error;

	const int compass_delay = 10; //mili-seconds to delay between compass
	                              //readings, this was 50
	const char init_str[] = "#BAD=8*7A\r\n"; //5 times per second, the first
	                                         //number was 11

	int string_pos;
	char input_char;

	unsigned long compass_wait_time;
	const int compass_timeout = 1;

	int Compass_update;
```

```
//Communication Setup

       word status, port;
       longword host;
       udp_Socket compass_data, gps_data, error_data, sonar_data;
       sock_type wp_data, man_data;
       char cmdBuf[1024];
       char cmdstr[20], *cmdptr;
       char wptBuf[4096];
       char wptstr[500], *wptptr, *wpttmp;
       char error_buf[200];

//Navigation Variables

       const float brg_error = 5.0;  //Allowable Bearing Error
       const float  rng_error = 5.0;  //Allowable range error (in yards)

       float lat_diff, lon_diff;   //The amount of Lat/Long (in Seconds and
                                   //Decimal Seconds between Bender's
                                   //current
                                   //position and the next waypoint
       float theta;              //Angle (deg) from True North to next waypoint
       float hdg_error;  //Angle (deg) from current heading to next waypoint

       float new_hdg;                //The Desired heading in degrees

       double rng, temp_rng;         //Range and temporary range (in yards)

//Waypoint Variables

       typedef struct
       {
              double lat;
              double lon;
              char  action;
       }WP;                             //Define WP structure

       WP waypoints[10];                //stores the list of waypoints
        char passed_waypoint[10];    //Stores action value for passed waypoints
       int curr_wp;                     //current wp
       char *temp;
       char *temp_lat, *temp_lon;
       char *temp_action;

       double lat, lon, wlat, wlon;

//     CTRL   bools

       int man_ctrl;
       int GPS_updated;

// Control Variables

    const float PW_MAX = 3.50;    //Max pulse width yields max fwd speed
    const float PW_STOP = 2.42;   //Pulse width that results in stop command
    const float PW_REV = 1.50;    //Pulse width that results in max reverse
    double Error;   //Variables for intermediate calculations of
    double sumError;                     //Running sum for integral error
    double prevError[10];       //Variable equal to error from previous n time
                                 //steps
    float lt_spd, rt_spd; //manual control variable, really could clean this up
```

```
    float spd; //Speed variable, need this for manual control, but not PID
    const int rt_ch = 0;       //right side
    const int lt_ch = 1;       //left side

//The new IIC functions and protocols

#ifndef i2c_SCL_H()
#define i2c_SCL_H()   BitWrPortI(PEDR,&PEDRShadow,0,0)
#define i2c_SCL_L()    BitWrPortI(PEDR,&PEDRShadow,1,0)
#define i2c_SDA_H()   BitWrPortI(PEDR,&PEDRShadow,0,1)
#define i2c_SDA_L()    BitWrPortI(PEDR,&PEDRShadow,1,1)
#endif
int i2c_clocks_per_us;

#define cWAIT_5_us  asm ld a, (i2c_clocks_per_us) $\
                sub 3 $\
          ld b,a $\
         db 0x10, -2

unsigned long t0;
#define time 5

//Collision Avoidance Variables

float fright, fleft, front, lside, rside, rear;

//////Begin the IIC protocol///////

void write_byte(char d)
{
       int i;
       for (i=0; i<8; i++)
       {
     for (t0=MS_TIMER;MS_TIMER<t0+time;);
              if (d & 0x80)
                     {i2c_SDA_H();}
              else
                     {i2c_SDA_L();}
       //cWAIT_5_us;

       i2c_SCL_H();
//     cWAIT_5_us;
   for (t0=MS_TIMER;MS_TIMER<t0+time;);
       i2c_SCL_L();
//     cWAIT_5_us;

       d=d<<1;
       }
   i2c_SCL_L();
   i2c_SDA_H();
}

int read_byte(char *ch)
{
   auto char res,cnt;
   i2c_SDA_H();

       for (cnt=0,res=0; cnt<8; cnt++)
       {
              i2c_SCL_H();
      while (BitRdPortI(PEDR,2)==0);//SCL Clock Stretching
//            cWAIT_5_us;
```

```
      for (t0=MS_TIMER;MS_TIMER<t0+time;);
      res<<=1;
             if(BitRdPortI(PEDR,3)) res|=0x01;
             i2c_SCL_L();
             //cWAIT_5_us;
      for (t0=MS_TIMER;MS_TIMER<t0+time;);
       }
       *ch=res;
   return 0;
}
void i2c_start_tx()
{

       i2c_SCL_H();
       i2c_SDA_H();
   cWAIT_5_us;
   i2c_SDA_L();
   cWAIT_5_us;
       i2c_SCL_L();

}


void i2c_stop_tx()
{

       i2c_SDA_L();
   for(t0=MS_TIMER;MS_TIMER<t0+time;);
   //cWAIT_5_us;
       i2c_SCL_H();
   cWAIT_5_us;
       i2c_SDA_H();
}


void i2c_init()
{
       int i;
       void i2c_stop_tx();
       i2c_SDA_H();
       cWAIT_5_us;
       i2c_SCL_L();
       for (i=0; i < 3; i++)
       {
             i2c_stop_tx();
       }
}


void giveack()
{
       i2c_SDA_L();
   cWAIT_5_us;
       i2c_SCL_H();
       for(t0=MS_TIMER;MS_TIMER<t0+200;);
   //cWAIT_5_us;
       i2c_SCL_L();
   cWAIT_5_us;
       i2c_SDA_H();
}
```

```
void getack()
{
       i2c_SDA_H();
       while (BitRdPortI(PEDR,3) == 1);
 if (BitRdPortI(PEDR,3) == 1)i2c_stop_tx();      //originally uncommented
       i2c_SCL_H();
   for (t0=MS_TIMER;MS_TIMER<t0+time;);
   //cWAIT_5_us;
       i2c_SCL_L();
}

///////End IIC and collision avoidance////////

// Function Prototypes

int compass_get_hdg(char sentence[MAX_SENTENCE]);

int gps_get_position(GPSPosition *newpos, char *sentence);

int gps_parse_coordinate(char *coord, int *degrees, float *minutes);

int ERROR_function(float new_hdg);

void msDelay (long sd);


///////Main Program///////

main()
{

       int i;

//Initialization

       char sonar;
   brdInit();
   i2c_init();


       //Communication Initialization

       sock_init();
       if (!(host = resolve(INTERFACE_ADDRESS))) {
              exit(3);
       }

       if (!udp_open(&error_data, ERROR_PORT, 0xffffffff, ERROR_PORT, NULL)) {
              exit(3);
       }
       sock_mode( &error_data, TCP_MODE_ASCII);
       sock_mode( &error_data, UDP_MODE_NOCHK);

       if (!udp_open(&wp_data, WP_PORT, 0xffffffff, WP_PORT, NULL)) {
              sock_puts(&error_data, "$Unable to open WP UDP session\n");
              exit(3);
       }
       sock_mode( &wp_data, UDP_MODE_NOCHK);

       if (!udp_open(&man_data, MAN_PORT, 0xffffffff, MAN_PORT, NULL)) {
              sock_puts(&error_data, "$Unable to open MANUAL UDP session\n");
              exit(3);
```

```
	}
	sock_mode( &man_data, UDP_MODE_NOCHK);

	if (!udp_open(&compass_data, COMPASS_PORT, 0xffffffff, COMPASS_PORT,
NULL)) {
		sock_puts(&error_data, "$Unable to open COMPASS UDP session\n");
		exit(3);
	}
	sock_mode( &compass_data, TCP_MODE_ASCII);
	sock_mode( &compass_data, UDP_MODE_NOCHK);

	if (!udp_open(&gps_data, GPS_PORT, 0xffffffff, GPS_PORT, NULL)) {
		sock_puts(&error_data, "$Unable to open GPS UDP session\n");
		exit(3);
	}
	sock_mode( &gps_data, TCP_MODE_ASCII);
	sock_mode( &gps_data, UDP_MODE_NOCHK);

	sock_puts(&error_data, "$Sockets are established\n");

	if (sock_recv_init( &wp_data, wptBuf, (word)sizeof(wptBuf))) {
		sock_puts(&error_data, "$Could not enable WP buffer.\n");
		exit(3);
	}
	if (sock_recv_init( &man_data, cmdBuf, (word)sizeof(cmdBuf))) {
		sock_puts(&error_data, "$Could not enable MAN buffer.\n");
		exit(3);
	}


	//Motor Initialization

       anaOutVolts(rt_ch, PW_STOP);
       anaOutVolts(lt_ch, PW_STOP);

	//Flag Initialization

		man_ctrl = 1;
		GPS_updated = 0;
		Compass_update = 0;

	//Detection Initialization

		time_flag = 0;
		detect_flag = 1;
		motion_flag = 0;
		a = 0;
		b = 0;

	//Compass Initialization

		serBopen(9600);  //BAUD rate
		serBwrFlush();
		serBputs(init_str);

	//GPS Initialization

		serCopen(9600);           // Open serial port C
		serCwrFlush();            // Flush serial port C Buffer
		serBputs(GPS_Reset);      // Send Reset signal to GPS Receiver
		serBputs(GPS_Sent_Clr);   // Send Clear signal to GPS Receiver
		serBputs(GPS_GGA_Enable); // Send GGA Sentence enable signal
```

```
                                //    (position info)

   //PID Initialization

            //printf("Enter new heading:  ");  //These are just for testing
            //new_hdg = atof(gets(s));
            //printf("Enter the desired Gain 1.0 to 1.3:  ");
            //P = atof(gets(p));
            P = 1.10;
            I = 0.05*P;
            D = 0.05*P;

       //Control Initialization

       sumError = 0.0;
       for (i = 0; i < 10; i++) prevError[i] = 0.0;

  //Turn the Brake on Initially

             digOut(0,1);

  ///////Main Loop///////

       while (1)
       {
             tcp_tick(NULL);

             ///////Recieve Manual Control Data

             costate
             {
                   waitfor(sock_recv( &man_data, cmdstr,
(word)sizeof(cmdstr)));

                   digOut(0,1);  //turn the brake on initially if it wasn't
already
                   detect_flag = 1;  //turn off the dectection
                   motion_flag = 0;  //reset the motion flag if AGV moves

                   //Tokenize the string and convert to integers
                   lt_spd = atof(strtok(cmdstr, " "));
                   rt_spd = atof(strtok(NULL, "/n"));

                   if ((rt_spd > 2.6) || (lt_spd > 2.6) || (rt_spd < 2.24) ||
(lt_spd < 2.24))
                         {
                         digOut(0,0);
                         detect_flag = 0;
                         } //turn off brake if get enough voltage

                   //Voltage to the motors (could clean this up)

                   anaOutVolts(rt_ch, rt_spd);
                   anaOutVolts(lt_ch, lt_spd);

                   if (!man_ctrl)
                   {
                         sprintf(error_buf, "$manual control data received, in
manual control, detection is off\n", curr_wp);
                         sock_puts(&error_data, error_buf);
                   }
```

```
                        //Update the flags

                        man_ctrl = 1;

                        //flush prevError array so next nav order will have minimum
error when it begins

                        for (i = 0; i < 10; i++) prevError[i] = 0.0;

                }//Recieve Manual Data


                ///////Compass

                costate
                {
                        waitfor ( DelayMs(compass_delay));

                        serBrdFlush();
                        string_pos = 0;

                        input_char = serBgetc();

                        //find begining of sentence

                        compass_wait_time = SEC_TIMER + compass_timeout;
                        //timeout if compass not working
                        while (input_char != '$')
                        {
                                if (SEC_TIMER > compass_wait_time) abort;
                                input_char = serBgetc();
                                //printf("%c",input_char);
                                msDelay(READDELAY);
                        }
                        //printf("\n");

                        //read the sentence

                        while (input_char != '*' )
                        {

                                        compass_sentence[string_pos] = input_char;
                                        string_pos++;
                                        if(string_pos == MAX_SENTENCE)
                                                string_pos = 0; //reset string large

                                input_char = serBgetc();
                                //printf("%c",input_char);
                                msDelay(READDELAY);
                        }

                        compass_sentence[string_pos] = 0; //add null
                        sock_puts(&compass_data, compass_sentence);
                        //tcp_tick(NULL);

                        if((compass_error = compass_get_hdg(compass_sentence)) !=0)
                        {
                        sprintf(error_buf, "$Compass Error: %d\n",compass_error);
                                sock_puts(&error_data, error_buf);
                                //tcp_tick(NULL);
        //printf("$Compass Error: %d\n %s\n",compass_error,compass_sentence);
                        }
```

```
                else
                {
                        //printf("Current heading: %f\n", curr_hdg);
                        Compass_update = 1;
                }

            //curr_hdg = 0.0;  //testing purposes
            }//Compass


            ///////Recieve WP Data

            costate
            {
            waitfor(sock_recv( &wp_data, wptstr, (word) sizeof(wptstr)));

                //find begining of string

                        wptptr = wptstr;          //assign a pointer
          while (*wptptr != '$')             //Step through until begin string
                        wptptr++;

                        wptptr++;

                //Tokenize

                temp_lat = strtok(wptptr, " ");
                temp_lon = strtok(NULL, " ");
                temp_action = strtok(NULL, " ");

            for (i = 0; i < 10; i++)
            {
                if ((temp_lat == 0 && temp_lon ==0) ||
                    waypoints[i].action != "P")
                {
                        waypoints[i].lat = strtod(temp_lat, NULL);
                            waypoints[i].lon = strtod(temp_lon, NULL);
                            waypoints[i].action = *temp_action;
                 //printf("wp%d: %f %f %c\n", i, waypoints[i].lat,
waypoints[i].lon, waypoints[i].action);
                            } //End if Statement

             temp_lat = strtok(NULL, " ");
                        temp_lon = strtok(NULL, " ");
                        temp_action = strtok(NULL, " ");
                           }//End for loop

                curr_wp = 0;  //Resets current WP to 1st waypoint. If this
is an update to
                      //waypoints, Nav will increment curr_wp until a good
                      //waypoint is there.

                //update the flags

                man_ctrl = 0;

                sprintf(error_buf,
                "$WP's recieved.  In AUTO NAV and preceeding to WP %d\n",
                curr_wp);
                sock_puts(&error_data, error_buf);
            }//End Waypoint Costatement
```

```
///////GPS

costate
{
       waitfor (DelaySec(gps_delay));
       serCrdFlush();
       string_pos = 0;
       input_char = serCgetc();

       //find begining of sentence

       //printf("\n");
       gps_wait_time = SEC_TIMER + gps_timeout; //timeout if gps
                                                  not sending data
       while (input_char != '$')
       {
              if (SEC_TIMER > gps_wait_time) abort;
              input_char = serCgetc();
              //printf("%c",input_char);
              msDelay(READDELAY);
       }

       while ((input_char != '\r') && (input_char !='\n'))
       {
              sentence[string_pos] = input_char;
              string_pos++;
              if(string_pos == MAX_SENTENCE)
                     string_pos = 0; //reset string if too large

              input_char = serCgetc();
              msDelay(READDELAY);
       }
       sentence[string_pos] = 0;
       sock_puts(&gps_data, sentence);
       //tcp_tick(NULL);
       gps_error = gps_get_position(&current_pos, sentence);
       if ((gps_error == 0) || (gps_error == -1))
              gps_error_count = 0;
       else
       {
              gps_error_count ++;

              //Stop AGV and place in manual control if BAD
                position data
  //          for 6 times (1 minute)
              if ((gps_error_count > 6) && man_ctrl == 0)
              {
                     sock_puts(&error_data,
        "$GPS error count exceeded. AGV in MANUAL CONTROL.\n");
                     tcp_tick(NULL);
                     digOut(0,1);

                     //update flags for manual control

                     man_ctrl = 1;
                     detect_flag = 1;
                     abort;     //still parse if -1
              }
       }

       if (1)//   (gps_error == 0)|| (gps_error == -1))
       {
```

```
                        GPS_updated = 1;
                        curr_lat=(current_pos.lat_degrees +
(current_pos.lat_minutes/60));
                        curr_lon=(current_pos.lon_degrees +
(current_pos.lon_minutes/60));

                  }


            }//GPS

      /***** Passes heading error and range to CTRL costatement
      * Nav**** and uses error function to determine error from
      ***** new_hdg and curr_heading*/

     costate
     {
                  if (man_ctrl) abort;

                  if (GPS_updated)    //Navigates to new waypoint
                  {
                        motion_flag = 0;  //reset the motion flag if AGV is
                                          //going to move

                  lat = 60 * curr_lat;  //converts latitude into
                                        //Minutes and decimal minutes
                   lon = 60 * curr_lon;            //converts longitude into
                                                 //Minutes and decimal minutes
            wlat = 60 * waypoints[curr_wp].lat;  //Converts waypoint values
            wlon = 60 * waypoints[curr_wp].lon;  //to decimal minutes
                              //printf("lat: %g\tlon: %g\n", lat, lon);
                              //printf("wlat: %g\twlon: %g\n", wlat, wlon);

            rng = sqrt((((2000 * wlat) - (2000 * lat)) * ((2000 * wlat) -
                  (2000 * lat))) + (((1600 * wlon) - (1600 * lon)) *
                  ((1600 * wlon) - (1600 * lon))));

            if (rng <= rng_error)  //When close enough to waypoint, action
                                   //code takes effect and next waypoint
                                   //is loaded

            {

             switch (waypoints[curr_wp].action)
                    {
                 case 'T':                //Go to next waypoint
                 {
                     passed_waypoint[curr_wp] = 'T';  //Stores action code
                                                      //in temp array
                     waypoints[curr_wp].action = 'P';
                       //Changes action code to indicate WP has been passed
                     sock_puts(&error_data, "$Proceeding to next WP\n");
                     curr_wp++;

                                   while ((waypoints[curr_wp].lat == 0) &&
                                          (waypoints[curr_wp].lon == 0))
                                     {
//checks for valid WP
                              curr_wp++;

                                          if (curr_wp == 10)
                                          {
```

```
            sock_puts(&error_data, "$No Valid WP Found\n");
               tcp_tick(NULL);
               man_ctrl = 1;
               detect_flag = 1;
               abort;
          }//End if
    }//End while

                    break;
              }//End case 'T'

case 'H':              //Start from beginning again
              {
     for (i = 0;i < 10;i++)   //Reloads prior action codes
     {
         waypoints[i].action = passed_waypoint[i];
     }
sock_puts(&error_data, "$Proceeding back to home WP. \n");
                  curr_wp = 0;

                  while ((waypoints[curr_wp].lat == 0) &&
                         (waypoints[curr_wp].lon == 0))
     {                 //checks for valid WP

                       curr_wp++;

                          if (curr_wp == 10)
         {
           sock_puts(&error_data, "$No Valid WP Found\n");
             tcp_tick(NULL);
             man_ctrl = 1;
             detect_flag = 1;
             abort;
          }//End if
             }//End while

                     break;
}//End case 'H'

case 'S':              //Stop
{
       digOut(0,1); //Stop AGV

      for (i = 0; i < 10; i++) //Clears the Waypoint array
    {
         waypoints[i].lat = 0;
         waypoints[i].lon = 0;
         waypoints[i].action='T';
     }//End for loop

     sock_puts(&error_data,
            "$Destination Achieved, Waypoints cleared\n");
     tcp_tick(NULL);
     man_ctrl = 1;
     detect_flag = 1;
     abort;
              }//End case 'S'

case 'C':  //Turn in a circle then proceed to next WP,
           //really don't use this
              {
     curr_wp++;
```

```
          while ((waypoints[curr_wp].lat == 0) &&
                                (waypoints[curr_wp].lon == 0))
                          {               //checks for valid WP
               curr_wp++;

                                 if (curr_wp == 10)
               {
                sock_puts(&error_data, "$No Valid WP Found\n");
                    tcp_tick(NULL);
                    man_ctrl = 1;
                    detect_flag = 1;
                    abort;
                }//End if
                  }//End while

                         break;
                }//End case 'C'

             case 'P':          //Check for passed waypoints
    {
      curr_wp++; //AGV ignores this point and goes to next one
          while ((waypoints[curr_wp].lat == 0) &&
                                (waypoints[curr_wp].lon == 0))
          {                 //checks for valid WP
               curr_wp++;

                                 if (curr_wp == 10)
               {
                sock_puts(&error_data, "$No Valid WP Found\n");
                   tcp_tick(NULL);
                   man_ctrl = 1;
                   detect_flag =1;
                   abort;
               }//End if
          }//End while

                         break;
                   }//End case 'P'

    default:            //Indicates and invalid action code
    {
 sprintf(error_buf, "$Invalid action for WP # %d\n", curr_wp);
                       sock_puts(&error_data, error_buf);
          tcp_tick(NULL);

          digOut(0,1);  //Stop AGV
          man_ctrl = 1; //in manual control
                                      detect_flag = 1;
          abort;
                  }//End default case
            }//End Switch

if (curr_wp > 9) //Action for last WP invalid.
{
                       digOut(0,1);  //Stop AGV
                       man_ctrl = 1;  //in manual control
                       detect_flag = 1;
     sock_puts(&error_data, "$Invalid action for wp 9\n");
     tcp_tick(NULL);
     abort;
           }//End if (curr_wp>9)
```

```
                }//End if (rng < rng_error)

              //If range not within error, calculate new heading

                // 3600 converts lat_diff and lon_diff to decimal seconds for
                // accuracy
                lat_diff = 3600 * (waypoints[curr_wp].lat-curr_lat);
                lon_diff = 3600 * (curr_lon - waypoints[curr_wp].lon);
                //printf("wp0_lat:  %g\twp0_lon: %g\n", waypoints[curr_wp].lat,
               //waypoints[curr_wp].lon);
                //printf("lat_diff: %g\tlon_diff: %g\n", lat_diff, lon_diff);

                // determine theta in degrees
                theta = atan((lat_diff) / (lon_diff)) * (180 / pi);
                //printf("theta: %g\n", theta);

                // waypoint located in positive y-axis
                if ((lon_diff == 0) && (lat_diff > 0))
                    new_hdg = 0;

                //waypoint is located in negative y-axis
                else if ((lon_diff == 0) && (lat_diff < 0))
                    new_hdg = 180;

                //waypoint is located in positive x-axis
                else if ((lon_diff > 0) && (lat_diff == 0))
                    new_hdg = 90;

                //waypoint is located in negative x-axis
                else if ((lon_diff < 0) && (lat_diff == 0))
                    new_hdg = 270;

                //waypoint is located in the first or fourth quadrant
                //(0-90 or 270-0)
                else if ((lon_diff > 0) && (lat_diff != 0))
                    new_hdg = 90 - theta;

                //waypoint is located in the second or third quadrant
                //(90-180 or 180-270)
                else if ((lon_diff < 0) && (lat_diff != 0))
                    new_hdg = 270-theta;

                hdg_error = ERROR_function(new_hdg);
                tcp_tick(NULL);

               }//End if (GPS_updated)
              }//End NAV costate


              ///////PID CONROL

              costate
              {
       waitfor(!man_ctrl);

      if ((hdg_error >= 180.0) || ((hdg_error > -180.0) && (hdg_error < 0.0)))
{flag = 1;}
                        else {flag = 0;}
                    if (hdg_error == 0.0) {flag = 2;} //just sets a dummy
number to flag

                    //calculate scale constant
```

```
                    compconv = fabs(hdg_error);
                    if (compconv > 180.0) {compconv = 360.0 - compconv;}
                    scale = ((compconv*(PW_MAX - PW_STOP))/180.0) + PW_STOP;
//related all of this to max and stop voltages, check this
                    scaleint = scale + scaleint;
                    if(flagint > 40){scaleint = 0.0;}
                    flagint++;

                                if ((!(hdg_error == 0.0)) && (rng >
                                rng_error))
                                {

                    //Translate this into an equivalent forward voltage
                    turnvolts = (P * scale) + (I * scaleint) + (D * scale);
                                //Do not send more than we put out
                                if(turnvolts > PW_MAX){turnvolts = PW_MAX;}
                                //Slower turn voltage for the other side
                                fwdturnvolts = 0.85*turnvolts;   //changed
                                //from 0.9 on 27 July, seems to work better
                                //Do not ever let the wheels go in reverse
                          if(fwdturnvolts < PW_STOP){fwdturnvolts = PW_STOP;}

                          //printf("The compass error is:  %f\n", hdg_error);
                    //printf("The forward voltage is:  %3.1f\n", turnvolts);
                    //printf("The slower voltage is:  %3.1f\n", fwdturnvolts);

                                detect_flag = 0;
                                digOut(0,0);

                                //turn logic
                                if(flag == 0)
                                      {
                                      anaOutVolts(rt_ch, fwdturnvolts);
                                      anaOutVolts(lt_ch, turnvolts);
                                      }
                                if(flag == 1)
                                      {
                                      anaOutVolts(rt_ch, turnvolts);
                                      anaOutVolts(lt_ch, fwdturnvolts);
                                      }
                                //if(hdg_error == 0.0)
                                      //{
                                      //printf("we made it\n");
                                      //anaOutVolts(rt_ch, PW_STOP);
                                      //anaOutVolts(lt_ch, PW_STOP);
                                      //}

                                }//ends if for heading error and range greater
                                //than range error

                                else
                                {
                                //send the right voltages to the wheels if no
//heading error and the range is greater than the delta

                                detect_flag = 0;
                                digOut(0,0);

                                anaOutVolts(rt_ch, PW_MAX);
                                anaOutVolts(lt_ch, PW_MAX);
                                }
```

```
        }//end PID costate

       ///////Collision Avoidance

       costate
              {

      waitfor(!man_ctrl);


       //Sonar Itself

              k = 0;
       sonarplus = 0;

   for (k=0; k<3; ++k)
       {
       i2c_start_tx();
       write_byte(0xE0);
       getack();
       write_byte(0x00);
       getack();
       write_byte(0x50);
       getack();
       cWAIT_5_us;
       i2c_stop_tx();
       for(t0=MS_TIMER;MS_TIMER<t0+65;);
       i2c_start_tx();
       write_byte(0xE0);
              getack();
       write_byte(0x03);
       getack();
       i2c_start_tx();
       write_byte(0xE1);
       getack();
       read_byte(&sonar);
       //printf("Sonar Inches: %d\n",read);
       i2c_stop_tx();
       if (sonar == 0)
              {
              sonar = 255;
              }  //this takes care of the sonar equal to zero when there is no
              //return, objects 20 plus feet away
              sonarplus = sonarplus + sonar;
              }
       sonarplus = (sonarplus/3);  //this averages three sonar readings 17 AUG


   //IR Rangers
   fright = anaInVolts(3);
   front = anaInVolts(4);
   fleft = anaInVolts(5);
   rear = anaInVolts(0);

   if ((front > 0.2 && front <= 2.0) || (fright > 0.2 && fright <= 2.0) ||
(fleft > 0.2  && fleft <= 2.0) || (sonarplus < 30 && sonarplus >= 10))
       {
       lside = anaInVolts(2);
       rside = anaInVolts(1);
       if (lside < rside)
       {
          anaOutVolts(lt_ch, PW_REV);
```

```
         anaOutVolts(rt_ch, PW_MAX);
         msDelay(500);
         anaOutVolts(lt_ch, (PW_MAX*0.82));
         anaOutVolts(rt_ch, (PW_MAX*0.82));
         }
      if (rside < lside)
       {
         anaOutVolts(rt_ch, PW_REV);
         anaOutVolts(lt_ch, PW_MAX);
         msDelay(500);
         anaOutVolts(lt_ch, (PW_MAX*0.82));
         anaOutVolts(rt_ch, (PW_MAX*0.82));
         }
             }
   else if ((front > 2.0 || fright > 2.0 || fleft > 2.0 || sonarplus < 10.0) &&
(rear < 1.0))
       {
      anaOutVolts(lt_ch, PW_REV);
      anaOutVolts(rt_ch, PW_REV);
      msDelay(1000);
      anaOutVolts(lt_ch, PW_STOP);
      anaOutVolts(rt_ch, PW_STOP);
      lside = anaInVolts(2);
       rside = anaInVolts(1);
      if (lside < rside)
       {
         anaOutVolts(lt_ch, PW_REV);
         anaOutVolts(rt_ch, PW_MAX);
         msDelay(500);
         anaOutVolts(lt_ch, (PW_MAX*0.82));
         anaOutVolts(rt_ch, (PW_MAX*0.82));
         }
      if (rside < lside)
       {
         anaOutVolts(rt_ch, PW_REV);
         anaOutVolts(lt_ch, PW_MAX);
         msDelay(500);
         anaOutVolts(lt_ch, (PW_MAX*0.82));
         anaOutVolts(rt_ch, (PW_MAX*0.82));
         }
      }

   //printf("Front IR = %f\n", front);
   //printf("Front Right IR = %f\n", fright);
   //printf("Front Left IR = %f\n", fleft);
   //printf("Left Side IR = %f\n", lside);
   //printf("Right Side IR = %f\n", rside);
   //printf("Rear IR = %f\n\n", rear);

   }   //Collision Avoidance Costatement


///////Detection Mode Costatement

       costate
             {
       if (detect_flag == 1)
                          {

                  digOut(0,1);
             motion1_volts = anaInVolts(6);
             motion2_volts = anaInVolts(7);
```

```
            //++time_flag;
                  //sprintf(error_buf, "$I'm in here %d times", time_flag);
                        //sock_puts(&error_data, error_buf);

                        //this filters out the false contacts as the motion
                        //detectors settle down

            if (motion1_volts >= 1.0) {++a;}
                        if (motion2_volts <= 1.0) {++b;}  //changed this on
                        //21 SEP, because of wiring
                        if (motion2_volts > 1.0) {b = 0;}

                        if (a >= 8)
                              {
                              ++motion_flag;
            sprintf(error_buf, "$Got Haji %d times (front)", motion_flag);
                              sock_puts(&error_data, error_buf);
                  a = 0;
                  }

                        if (b >= 6)
                              {
                              ++motion_flag;
            sprintf(error_buf, "$Got Haji %d times (side)", motion_flag);
                              sock_puts(&error_data, error_buf);
                  b = 0;
                  }
            } //end if
      else {abort;}

       }  //end detection costate



       }//while(1)


}//main


////////////////////////////////////////////////////////////////


/* START FUNCTION DESCRIPTION ****************************************
compass_get_hdg

SYNTAX:      int compass_get_data();

KEYWORDS:      compass

DESCRIPTION:   Parses a sentence to extract heading data.
             This function is able to parse HPR data from a
             HMR3000 Digital Compass

PARAMETER1:  sentence - a string containing a line of HPR data

RETURN VALUE:        0 - success
             -1 - parsing error

             -2 - heading marked invalid
```

```
SEE ALSO:

END DESCRIPTION ************************************************************/

int compass_get_hdg(char sentence[MAX_SENTENCE])
{
       auto int i;
       char *err,*hdg,*type;
       char error;

       if(strlen(sentence) < 4)
              return -1;
       if(strncmp(sentence, "$PTNTHPR", 8) == 0)
       {
              //parse hpr sentence
              type = strtok(sentence, ",");
              hdg = strtok(NULL, ",");
              err = strtok (NULL, ",");
              if(hdg == NULL)
                     return -2;

              //pull out data
              curr_hdg = atof(hdg);

              error = (int)err;
              if (strncmp(&error, "N", 1) == 0)
                     return -2;

       }
       else
              return -1;

       return 0;

}

/* START FUNCTION DESCRIPTION ****************************************
gps_parse_coordinate

SYNTAX:      gps_parse_coordinate(char *coord, int *degrees, float *minutes)

KEYWORDS:      gps parse

DESCRIPTION:  Parses GPS position data

PARAMETER1:  coord - contains N/S, E/W
            degrees, minutes - positional information

RETURN VALUE:        0 - success (xxxxx.xxxx minutes)
              -1 - parsing error

SEE ALSO:

END DESCRIPTION ************************************************************/

nodebug int gps_parse_coordinate(char *coord, int *degrees, float *minutes)
{
       auto char *decimal_point;
       auto char temp;
       auto char *dummy;

       decimal_point = strchr(coord, '.');
```

```
	if(decimal_point == NULL)
		return -1;
	temp = *(decimal_point - 2);
	*(decimal_point - 2) = 0; //temporary terminator
	*degrees = atoi(coord);
	*(decimal_point - 2) = temp; //reinstate character
	*minutes = strtod(decimal_point - 2, &dummy);
	return 0;
}


/* START FUNCTION DESCRIPTION ****************************************
gps_get_position

SYNTAX:			int gps_get_position(GPSPositon *newpos, char
*sentence);

KEYWORDS:	gps

DESCRIPTION:	Parses a sentence to extract position data.
				This function is able to parse any of the
following
				GPS sentence formats: GGA

PARAMETER1:	newpos - a GPSPosition structure to fill
PARAMETER2:		sentence - a string containing a line of GPS data
				in NMEA-0183 format

RETURN VALUE:	0 - success
				-1 - not differential
				-2 - sentence marked invalid
				-3 - parsing error

SEE ALSO:

END DESCRIPTION ************************************************************/

//can parse GGA
nodebug int gps_get_position(GPSPosition *newpos, char *sentence)
{
	auto int i;

	if(strlen(sentence) < 4)
		return -3;
	if(strncmp(sentence, "$GPGGA", 6) == 0)
	{
		//parse GGA sentence
		for(i = 0;i < 11;i++)
		{
			sentence = strchr(sentence, ',');
			if(sentence == NULL)
				return -3;
			sentence++; //first character in field
			//pull out data
			if(i == 1) //latitude
			{
				if( gps_parse_coordinate(sentence,

&newpos->lat_degrees,

&newpos->lat_minutes)
				)
```

```
                        {
                                return -3; //get_coordinate failed
                        }
                }
                if(i == 2) //lat direction
                {
                        newpos->lat_direction = *sentence;
                }
                if(i == 3) // longitude
                {
                        if( gps_parse_coordinate(sentence,

&newpos->lon_degrees,

&newpos->lon_minutes)
                          )
                        {
                                return -3; //get_coordinate failed
                        }
                }
                if(i == 4) //lon direction
                {
                        newpos->lon_direction = *sentence;
                }
                if(i == 5) //link quality
                {
                        if(*sentence == '0')
                                return -2;
                        if(*sentence == '1')
                                return -1;
                }
        }
    }
    else
    {
        return -3; //unknown sentence type
    }
    return 0;
}

/* START FUNCTION DESCRIPTION ************************************
ERROR_function

SYNTAX:          int ERROR_function(new_hdg);

KEYWORDS:       nav, control

DESCRIPTION:    Determines heading error for use by Nav and Control
costatements,
                                currently allows a 6 degree range

PARAMETER1:     new_hdg - latest update of bearing to next waypoint or direction
                     to drive based upon collision contact

RETURN VALUE:   hdg_error

SEE ALSO:

END DESCRIPTION ***********************************************************/
int ERROR_function(float new_hdg)
{
    hdg_error = new_hdg - curr_hdg;
```

```
    if (hdg_error <= 6.0 && hdg_error > 0.0)
                    {
                    hdg_error = 0.0;
                    }
        if (hdg_error >= -6.0 && hdg_error < 0.0)
                    {
                    hdg_error = 0.0;
                    }

        return(hdg_error);
}

/* START FUNCTION DESCRIPTION ********************************************
gps_get_position

SYNTAX:                   void msDelay(long sd);

KEYWORDS:      delay, wait

DESCRIPTION:   introduces a defined ms delay loop

PARAMETER1:    sd - number of ms to wait

SEE ALSO:

END DESCRIPTION **********************************************************/
void msDelay (long sd)
{
       unsigned long t1;

       t1 = MS_TIMER;
       for (t1 = MS_TIMER; MS_TIMER < (sd + t1); );
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] ARMY TECHNOLOGY. Hunter RQ-5A. http://www.army-technology.com/project_printable.asp?ProjectID=2355, August 2006.

[2] FOSTER-MILLER. Talon Robots. http://www.foster-miller.com/lemming.htm, August 2006.

[3] DUNBAR, T. Demonstration of waypoint navigation for a semi-autonomous prototype surf-zone robot. Master's Thesis, Naval Postgraduate School, June 2006.

[4] SUPERDROID ROBOTS. Mini ATR. http://www.superdroidrobots.com/ATR_mini.htm, August 2006.

[5] SUPERDROID ROBOTS. Battery Pack. http://www.superdroidrobots.com/shop/item.asp?itemid=350, August 2006.

[6] NTE ELECTRONICS INC. NTE933. Voltage Regulator Specification Sheet.

[7] SUPERDROID ROBOTS. RPM Gear Motor. http://www.superdroidrobots.com/shop/item.asp?itemid=373, August 2006.

[8] SUPERDROID ROBOTS. Motor Controller. http://www.superdroidrobots.com/product_info/PWM_9_14_05.pdf, August 2006.

[9] SUPERDROID ROBOTS. SRF08. http://www.superdroidrobots.com/product_info/SRF08.htm, August 2006.

[10] FRADEN, J. Handbook of Modern Sensors, 3rd Ed., Springer-Verlag New York, Inc. 2004, pp. 286-288.

[11] SUPERDROID ROBOTS. GP2D12. http://www.superdroidrobots.com/product_info/SharpGP2D12-15.pdf, August 2006.

[12] DIY KIT 30. PIR Detector. http://kitsrus.com/pdf/k30.pdf, August 2006.

[13] FRADEN, J. Handbook of Modern Sensors, 3rd Ed., Springer-Verlag New York, Inc. 2004, pp. 245-248.

[14] DIY KIT 49. Ultrasonic Movement Detector. http://kitsrus.com/projects/k49.pdf, August 2006.

[15] Z-WORLD. BL2000. http://www.zworld.com/products/bl2000/, August 2006.

[16] HONEYWELL. HMR3000. http://www.ssec.honeywell.com/magnetic/datasheets/hmr3000_manual.pdf, August 2006.

[17] GARMIN. WAAS. http://www.garmin.com/about/GPS/waas.html, August 2006.

[18] GARMIN. GPS 16. http://www.garmin.com/manuals/GPS16HVS_TechnicalSpecifications.pdf, August 2006.

[19] D-Link. DCS-900. http://www.dlink.com/products/resource.asp?pid=270&rid=807&sec=0, October 2006.

[20] TOSCANO, M. Department of Defense Joint Robotics Program. *International Conference on Unmanned Ground Vehicle Technology* (April 2000).

[21] UZAN, K. SE4015 Class Presentation (June 2003).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Physics Department
   Naval Postgraduate School
   Monterey, California

4. Dr. Richard Harkins
   Department of Applied Physics
   Naval Postgraduate School
   Monterey, California

5. Dr. Nancy Haegel
   Department of Applied Physics
   Naval Postgraduate School
   Monterey, California